An Efficient Parallel IP Lookup Technique for IPv6 Routers Using Multiple Hashing with Ternary marker storage

P. Kiran Sree[†] Dr. Inampudi Ramesh Babu[††]

[†] Mr. P.Kiran Sree,Associate Professor, Department of Computer Science, S.R.K Institute of Technology, Enikepadu, Vijayawada,India, pkiransree@gmail.com, Mobile: +919959818274.

[††] Dr. Inampudi Ramesh Babu,Senior IEEE member & Professor, Department of Computer Science, Acharya Nagarjuna University,Guntur,India.

Abstract

Internet address lookup is a challenging problem because of the increasing routing table sizes, increased traffic, higher speed links, and the migration to 128 bit IPv6 addresses. Routing lookup involves computation of best matching prefix for which existing solutions scale poorly when traffic in the router increases or when employed for IPV6 address lookup. Our paper describes a novel approach which employs multiple hashing on reduced number of hash tables on which ternary search on levels is applied in parallel. This scheme handles large number of prefixes generated by controlled prefix expansion by reducing collision and distributing load fairly in the hash buckets thus providing faster worst case and average case lookups. The approach we describe is fast, simple, scalable, parallelizable, and flexible.

Keywords: IPv6 address lookup, routing, multiple hashing, controlled prefix expansion.

1. Introduction

Rapid growth of the Internet has created a demand for high speed routers that can handle very large routing tables with high data rate. Internet traffic is doubling every three months partly because of increased users, and also because of new multimedia applications. The higher bandwidth need requires faster communication links and faster network routers. Thus the key to improved Internet performance is faster routers. IP lookup is one of the central bottlenecks in router forwarding.

Moreover, the destination address of an arriving packet does not carry with it the information needed to determine the length of the longest matching prefix. Hence, we cannot find the longest match using an exact match search algorithm. Instead, a search for the longest matching prefix needs to determine both the length of the longest matching prefix as well as the forwarding table entry containing the prefix of this length that matches the incoming packet's destination address, which makes the lookup problem even more complex.

Most of the IP address lookup schemes developed so far, both hardware and software, have dealt with the IPv4 routing mechanism efficiently to a large extent. However, with the deployment of the IPv6 routing protocol (address length = 128 bits) the problem of routing millions of communication packets every second based on longest prefix match becomes cumbersome.

The standard approach used by an IP router to forward a packet is to keep a forwarding table based on IP destination address prefixes. Each prefix is associated with the next hop towards the destination. The IP router looks in its table for the longest prefix that matches the destination address of the packet, and forwards according to that match.

The rest of the paper is organized as follows. Section 2 describes existing approaches for IP lookups in a broad sense and highlights their limitations. Section 3 describes our new approach of applying multiple hashing on tables with prefixes which are expanded by CPE. It also describes an improved parallel ternary

search. Section 4 analyses the performance of the approach. We conclude Section 5 by assessing the contributions of this paper.

2. Related Research

The Software based IP routing lookup schemes introduced so far can be broadly classified into three categories, viz., Trie based schemes, range search schemes and hash based schemes [6].

2.1 Trie Based Schemes

Trie based schemes include BSD UNIX implementation of Patricia trie [5], multi-ary trie with controlled prefix expansion [3], Level compressed trie [6], and Lulea algorithm [6]. Basically most of these trie based schemes have a worst case lookup complexity proportional to the address length. Furthermore, a significant amount of storage space is wasted in a (radix) trie in the form of pointers that are null, and that are on *chain (*paths with 1-degree nodes, i.e., that have only one child). Thus the implementation can require up to 32 or 128 worst case costly memory accesses for IPv4 and IPv6 respectively. Hence scalability is poor for the IPv6 address lookup problem. Even in the best case, with binary branching and 40,000 prefixes, trie implementations can take $\log_2(40{,}000) = 16$ memory accesses.

2.2 Range Search Scheme

The range search scheme gets rid of the length dimension of prefixes and performs a search based on endpoints of addresses [6]. The number of basic intervals in the worst case is 2N, where N is the number of prefixes in the lookup table). Also, the best matching prefix (BMP) must be pre-computed for each basic interval, and in the worst case an update needs to re-compute the BMP of N basic intervals. The update complexity is O (N).

2.3 Binary Search on Levels

One attack for solving longest matching prefix problem is to perform binary search on levels [6]. Prefixes are divided according to length, with all prefixes of a given length in a table. We then perform a binary search for matching prefixes of the destination address according to the prefix lengths. Tables for each prefix length can be stored as a hash table. So, if there are W different possible prefix lengths, the search requires O ($\log_2$W) time.

2.4 Controlled prefix expansion

Controlled prefix expansion is a technique which converts a set of prefixes with M distinct lengths to a set of prefixes with a smaller number of distinct lengths. So if the number of distinct prefix lengths is l and l<W, then we can refine the worst case bounds for tries and binary search on levels to O (l) and O (log2l) respectively. It also reduces number of markers significantly since number of levels is reduced. Markers are added in the respective tables. Markers for each prefix are stored in all the potential levels that would be reached during the lookup. Also markers are inserted only at the levels whose length is smaller than that of the prefix to be searched or inserted [4].

But expansion of prefixes leads to increase in number of entries in the forwarding table, and there by leads to collision of entries in the hash tables when used with binary search on levels.

3. PROPOSED WORK

Worst case lookups in the case of trie based schemes depend on the maximum prefix length or address length which is too large for IPv6. Moreover optimizations based on number of prefixes or entries in the lookup table would not scale well in the future since the Internet traffic is increasing significantly. It instantly directs towards optimizations based on prefix lengths which are likely to be more robust for IPv6 addresses. This suggests the use of controlled prefix expansion to reduce the number of distinct prefix lengths to k, where k<l. If we can reduce the worst case lookups from 7 to 3 or 4 we can double the performance.

But the increase in number of prefixes due to controlled prefix expansion causes collision in the hash tables which if not handled will remain a hindrance to acquiring high throughput in route lookup. So we propose an approach in which we use multiple hashing to store prefixes in hash tables.

So the combined effect of Internet traffic surge and increase in the number of prefixes due to expansion makes handling collision a very important task. So the use of multiple hashing [2], distributes load in the buckets fairly and also reduces collision in hash tables and thereby

leveraging the benefits of controlled prefix expansion.

In particular, we emphasize that by properly structuring the data, one can parallelize memory accesses so that using multiple hash functions is desirable. One of the first analysis suggests using multiple tables, with a separate hash function for each table. Elements that collide in one table percolate to the next.

Our hash table consists of n buckets. We split the n buckets into two disjoint equal parts, which for convenience we call the left and the right. When an item is inserted, we call both hash functions, where each hash function has a range of [1,n\2]. The first hash function determines a bucket on the left, the second a bucket on the right. The item is placed in the bucket with smallest number of existing items; in the case of a tie, the item is placed in the bucket on the left. The main point here is that two hash functions allow greater predictability and a smaller maximum load, even while using much less memory.

The lookups are independent and can be done in parallel. As the leftmost groups are more likely to hold more items, they can be examined first. If the pattern is found in one hash bucket, the other need not be checked. Further, using multiple hash functions can dramatically improve upon the total space used to store the hash table by reducing the amount of unused space. We need to tradeoff the effectiveness and the computational cost of the hash functions, thus CRC is an optimal choice. This scheme is designed to solve the problems introduced by searching for a semi-perfect hash function, by instead using multiple hash functions. This approach is very general and proves highly suitable for IPv6 address lookup problem.

Markers are added for all prefixes to guide the search in the appropriate direction. Finding a marker implies that a larger best-matching-prefix may be found in the right or in the higher levels, whereas absence of marker indicates that there can be no BMP on the right or in higher levels of the table. A naive view would indicate placing a marker for prefix P at all levels in L higher than the level of P. However, it suffices to place markers at all levels in L that could be visited by binary or ternary search when looking for an entry whose BMP is P.

Consider an example where there are 53 hash tables, we can determine the potential tables in which markers are supposed to be added for the prefixes in 53^{rd} table, H_{53} to guide parallel ternary search on levels. First, the searching algorithm looks up in the 18^{th} and 36^{th} tables; hence we add markers in those tables so that the Lookup algorithm will decide to move to part3 of the levels. Subsequently, it can proceed to search from 37^{th} table to 53^{rd} table. Now it decides to lookup in the 42^{nd} and 48^{th} tables thus we add markers there so that it will direct the search from 49^{th} table to 53^{rd} table. So now the algorithm looks up in 50^{th} and 52^{nd} tables where again we need to add markers. Finally the algorithm finds the BMP in table 53. Note that we can have as many as 6 markers for a prefix in the worst case.

3.1 Pseudo code for storing markers:

Considering there are N tables each of distinct prefix lengths as follows:

Tables:	H_1,	H_2,	H_3	.	.	. H_n
Prefix Lengths:	l_1,	l_2,	l_3	.	.	. l_n

```
marker[6];  // array

Function Ternary_marker_storage
  For each table Hi of distinct prefix length li
    L=1, H=N, x=0;
    x = marker_storage (x, i, L, H);
        For each prefix p of the table Hi of
        distinct prefix length li
          For temp ← 0 to x-1
                Store marker of length
                l_marker[temp] in the table
                H_marker[temp] corresponding to
                prefix p of table Hi.
          End for
        End for
    End for
End Function

Function marker_storage(x, i, L, H)
        If(L==H)
                return x;
        Else
          A=(2L+H-1)/3;
          B=(L+2H+1)/3;
          If( (i==A) || (i==B) )
             return x;
          If( (i<A)&&(i<B) )
              marker_storage(x, i, L, A-1);
          Else if( (i>A)&&(i<B) )
              marker[x]=A;
              x=x+1;
              marker_storage(x, i, A+1, B-1)
          Else if( (i>A)&&(i>B) )
               marker[x]=A;
```

```
            x++;
            marker[x]=B;
            x++;
            marker_storage(x, i, B+1, H);
        End if
    End if
End Function
```

Then Parallel Ternary Search on levels is employed which splits the levels to be searched into three parts. It performs two lookups corresponding to the two boundaries of the three parts in parallel and then determines which part of the levels is likely to contain a BMP. The search is guided by markers as shown in fig. 1.

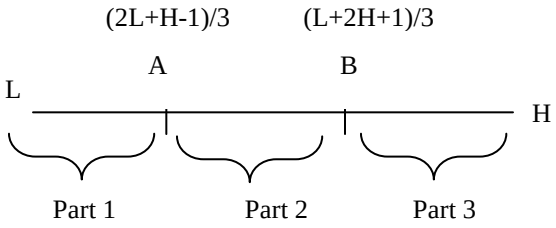

Fig. 1 Parallel Ternary Search

Thus parallel ternary search performs parallel lookup at levels A and B and determines which part (1, 2, and 3) contains the actual BMP and calls parallel Ternary search of the corresponding part recursively.

```
Parallel_Ternary_Lookup(L,H)
{
    if(L==H)
    return BMP;
    else
    {
        A=Lookup((2L+H-1)/3)
        B=Lookup((L+2H+1)/3);
        // actions are taken according
        to the rules specified in table 1.
    }
}
```

P Prefix found
M Marker found
PM Prefix and Marker found
\- Nothing found

Valid bit =0 indicates that such a combination is not possible.

When a matching prefix is found then it is stored as the BMP; if a longer matching prefix is found later then it replaces the current BMP. A marker indicates that a longer matching prefix can be found for the given address though it does not ensure that a better match will be found. So at times a marker can be misleading, such markers are called false markers. For instance say the address length is 4 bits, a marker for a prefix 0001* found in the second table 00* matches an address 0010 though the prefix corresponding to the marker does not match the address. So the marker in this case is misleading and hence it is a false marker.

Case	A	B	Valid	Action – handles false markers
1	P	P	0	-
2	P	M	0	-
3	P	PM	0	-
4	P	-	1	Store A'p as BMP
5	M	P	1	Store B'p as BMP
6	M	M	1	Lookup in part3 Lookup in part1 Lookup in part2
7	M	PM	1	Store B'p as BMP Lookup in part3
8	M	-	1	Lookup in part2 Lookup in part1
9	PM	P	1	Store B'p as BMP
10	PM	M	1	Store A'p as BMP Lookup in part3 Lookup in part2
11	PM	PM	1	Store B'p as BMP Lookup in part3
12	PM	-	1	Store A'p as BMP Lookup in part2
13	-	P	1	Store B'p as BMP
14	-	M	0	-
15	-	PM	0	-
16	-	-	1	Lookup in part1

Table 1

Parallel Ternary Search also handles the cases of false markers. For instance consider the case 8 in which we have a marker which may be a false marker so we need to lookup for the best matching prefix in part1 also.

4. Performance Evaluation

Reducing number of levels of hash tables improves worst case performance in both binary and ternary search on levels. The following

graphs depict the number of lookups for all prefix lengths with and without prefix expansion for binary search (fig. 2) and ternary search (fig.3) respectively.

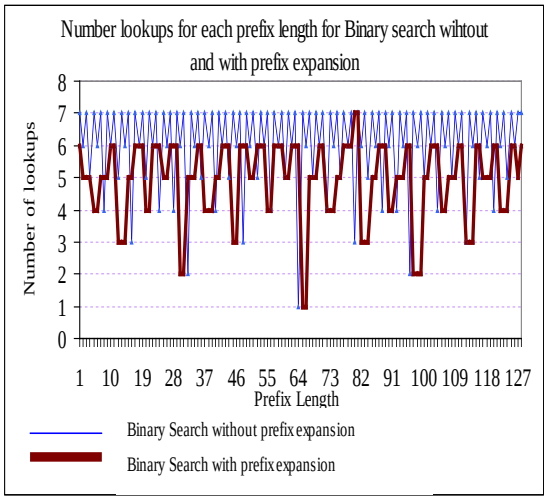

fig. 2

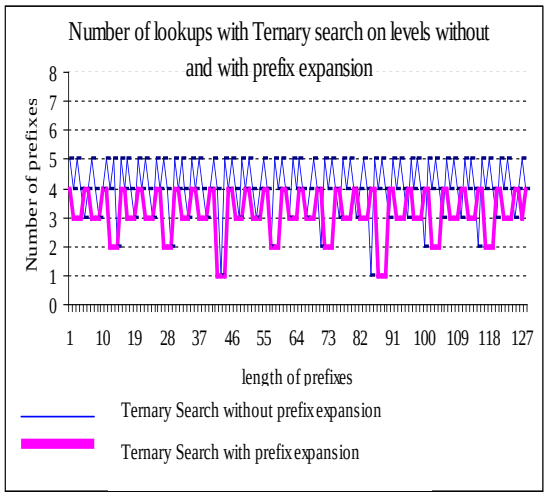

fig. 3

The improved parallel ternary search technique gives faster worst case (4 lookups) and average case lookups (3 lookups) than binary search on levels (worst case – 6 and average case – 5 lookups). The graph below (fig.4) shows the number of lookups for all prefix lengths for binary and parallel ternary search on levels.

We choose the number of distinct prefix lengths for ternary search based on natural heuristics using the expression

$X_n = 3 * X_{n-1} + 2$, where $X_0=1$

Thus we can choose to have 5, 17 and 53 as options for number of distinct prefix lengths. Using 53 hash tables will lead to 4 lookups in the worst case, and having 17 hash tables lead to 3 worst case lookups and having 5 hash tables to 2 lookups.

Using 53 hash tables each corresponding to one prefix length, with appropriate markers the number of lookups for prefixes of different lengths in binary and ternary search on levels is given by the figure, fig. 4.

Number of distinct prefix lengths = 53;

Distinct prefix lengths allowed are:

{1, 4, 6, 9, 11, 14, 16, 19, 21, 24, 26, 29, 31, 34, 36, 39, 41, 44, 46, 49, 51, 54, 56, 59, 61, 64, 66, 69, 71, 74, 76, 79, 81, 84, 86, 89, 91, 94, 96, 99, 101, 104, 106, 109, 111, 114, 116, 119, 121, 124, 126, 127, 128}

Further we provide a simple approximate fluid limit analysis of the multiple hashing schemes. Considering the case with d=2, where we have two groups and n/2 buckets, the fraction of the n hash buckets that contain atleast i items is
$X_i(1) \leq (2^{-F_d^{(i)}})/ d$

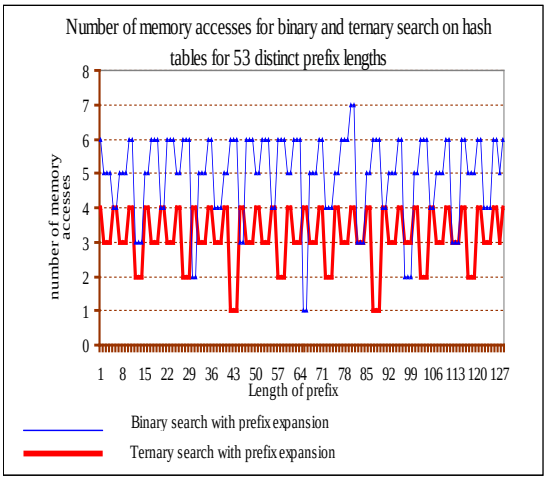

fig. 4

Intuitively, this implies that the x_i fall extremely quickly with i, and hence the maximum load is very small.

5. Conclusion

We have proposed a new approach for best matching search. The combination of multiple hashing and prefix expansion schemes proposed are suitable for situations where it is important to limit the maximum number of items that fall into a bucket. The key feature is that all hashes and memory lookups can be done in parallel. The improved parallel ternary search provides faster worst-case and average-case lookup times.

6. Acknowledgement

I am grateful to Dr. P. Venkata Narasaiah Director, S.R.K Institute of Engineering and Technology, for his consistent support at every stage of my research work. I specially thank our President Sri B.S Apparao & Secretary Sri B.S. Sri Krishna for providing necessary infrastructure. I am indebted to my colleagues for providing wonderful atmosphere to work with.

7. References

[1] *T. Srinivasan et al.* "High Performance Parallel IP Lookup Technique Using Distributed Memory Organization" *In Proc. of the IEEE International Conference of Information Technology (ITCC'04)*, Las Vegas, USA.

[2] Andrei Broder, Michael Mitzenmacher. "Using Multiple Hash Functions to improve IP Lookups" *IEEE INFOCOM,* 2001.

[3] *V. Srinivasan, G. Varghese.* "Fast address lookups using controlled prefix version". *ACM Transactions on Computer Systems*, Vol. 17, No. 1, February 1999, Pages 1–40.

[4] *M. Wald Vogel, G. Varghese, J. Turner, B. Plattener*, "Scalable High Speed IP Routing Lookups". *In Proc. Of SIGCOMM* 97, 1997.

[5] *Miguel Á. Ruiz-Sánchez, INRIA Sophia Antipolis, Universidad Autónoma etropolitana Ernst W. Biersack, Institut Eurécom Sophia Antipolis Walid Dabbous, INRIA Sophia Antipolis,.*"Survey and Taxonomy of IP Address Lookup Algorithms*"*. *IEEE Network* • March/April 2001

[6] *Pankaj Gupta* "*Algorithms* For Routing Lookups and Packet Classification" December 2000.

P.KIRAN SREE received his **B.Tech** in Computer Science & Engineering, from J.N.T.U and **M.E** in Computer Science & Engineering from Anna University. He has published many technical papers; both in international and national Journals .His areas of interests include Parallel Algorithms, Artificial Intelligence, Compile Design and Computer Networks. He also wrote books on Analysis of Algorithms, Theory of Computation and Artificial Intelligence. He is now associated with S.R.K Institute of Technology, Vijayawada.