A Dynamic Binary Hash Scheme for IPv6 Lookup

Qiong Sun[1] XiaoHong Huang[1] XiaoJu Zhou[1] Yan Ma[1,2]
1. School of Computer Science and Technology 2. Beijing Key Laboratory of Intelligent Telecommunications Software and Multimedia
Beijing University of Posts and Telecommunications, Beijing, 100876, P.R.China
{sunq,huangxh,zhouxj }@buptnet.edu.cn mayan@bupt.edu.cn

Abstract—**Recently, the significantly increased IPv6 address length has posed a greater challenge on wire-speed router for IP lookup. As a result, even the most efficient IPv4 lookup scheme can not meet the demand in IPv6.**

In this paper, we make a thorough observation on the characteristic of IPv4/IPv6 routing table and propose a novel division technique for routing table, with which the traditional LPM (longest prefix matching) problem can be changed to an exact matching one for majority prefixes. Based on this technique, we propose a DBH (dynamic binary hash) scheme for dynamic IPv6 routing lookup, which achieves O(log*W*) *(W* stands for the address length) lookup performance. Key feature of the new scheme is its ability to have a performance guarantee of 7 hash probes in the worst cast of IPv6 routing lookup, meanwhile it can support incremental update. This makes it more attractive in reality.

The performance evaluation has shown that the newly proposed scheme can achieve an average lookup memory access number of 2.7 for current IPv6 lookup. It is below 35% of best existing dynamic algorithm. Besides, the memory cost by the data structure itself is extremely small and the average update memory access number is very few.

Keywords-**routing lookup; dynamic hash scheme; maximum disjoint set**

I. INTRODUCTION

Nowadays, the next generation IPv6 has been gaining wider acceptance to replace its predecessor, IPv4. Although IPv6 can provide extremely large pool of address space for the Internet, it also poses a great challenge on the data path functions of IP lookup and packet classification in a router.

Usually, a router classifies incoming packets into flows utilizing information contained in packet headers and a table of rules. Actually, the IP lookup problem is a special case of packet classification problem when matching is usually performed on simply destination address. And most of the proposed packet classification schemes transform a multi-dimension lookup problem into a single-dimension one[1][2]. Thus, a good IP lookup solution can also be used to solve packet classification problem. This is the direction we pursue in this proposal.

In this paper, we mainly concentrate on one-dimension IP lookup problem and its implementation, in which LPM is the most complicated part. Generally, the metrics taken into consideration for a lookup algorithm are usually lookup speed, update time, memory cost and scalability. We believe that worst case lookup speed is as important as average lookup speed for a backbone router since it can guarantee the overall performance. For update speed, it has been noticed that backbone protocols can cause frequent prefix insertion or deletion (more than 1000 per second). Besides, dynamic SLA (service level agreement) has made ACL (access control list) to be a dynamic one. So in reality, both one-dimension and multi-dimension lookup scheme that can not support incremental update is not practical. In addition, memory cost and scalability for future use is also very important to IPv6.

In this paper, we make a detailed observation on the characteristic of MDS (Maximum Disjoint Set) of IPv4/ IPv6 routing tables and find out that that routing table is disjoint-prevalent. With this observation, we propose a novel division technique for the routing table and adopt hash data structures for our IPv6 lookup scheme, which is called DBH. Our scheme can not only have lookup performance of only O(log*W*), in which at most 7 hash probes will be taken for both current and future routing table, but also achieve dynamic property. This is a significant improvement to the static O(log*W*) scheme proposed by Waldvogel[3], since no one has ever solved the dynamic problem in [3] and achieve the lookup performance of O(log*W*) in the same time. In addition, IPv6 prefix distribution characteristic is taken into consideration to optimize the searching path.

The remainder of this paper is organized as follows. In Section II, previous related work will be presented. In Section III, a novel division technique for the routing table will be proposed. In Section IV, our proposed scheme DBH will be described, including detailed lookup, inserting and deleting process. In Section V, the experimental results of the proposed scheme and performance comparison with related algorithms are presented. Finally, Section VI concludes the paper with a summary.

II. PREVIOUS RELATED WORK

A. *Schemes for IPv4 routing lookup*

Currently, there are many lookup algorithms proposed for IPv4. They can be classified into trie-based, range-based and hash-based algorithms. These algorithms can be implemented on software, hardware or both. Software schemes can benefit from low cost and flexible. Hardware solutions, e.g. TCAM, can search contents in parallel and be updateable with recent development [4]; however, they still suffer from limited memory and high power consumption. Thus, only limited entries are suitable to perform in TCAM.

Project 60772111 supported by National Natural Science Foundation of China

- trie-based algorithm

The original one-bit trie algorithm organizes the prefixes with the bits of prefixes to direct the branching. However, since it's worst case memory accesses can be so high that many other techniques have been integrated to reduce the height of trie (like Patricia[5], LC trie[6], LPFST[7] etc), but the usage of these techniques often lead to hard updating or memory expansion.

- range-based algorithm

As a prefix by its nature can be represented by the start point and end point, lots of algorithms [8][9] have been introduced to search these endpoints. These algorithms can usually have O(log*N*) (*N* stands for the number of entries in a routing table) performance and can support incremental update.

- hash-based algorithm

Hash has an outstanding feature of O(1) run-time, and many network processors have build-in hash units. Besides, lots of research works have proposed a great many perfect hash functions for routing lookup. Waldvogel in[3] proposed a BSOL (Binary Search On Length) scheme of O(log*W*) performance. It's lookup performance is excellent and it has also attracted lots of attention in multi-dimension field [1][2]. Nevertheless, BSOL is a static algorithm, in which all prefixes' BMP (best matching prefix) need to be re-computed when updating one entry.

B. Schemes for IPv6 routing lookup

Up to now, there are very few algorithms designed dedicatedly for IPv6. Although many IPv4 algorithms can be simply upgraded to IPv6, their performance can not meet the requirement in IPv6 due to the high computation complexity of O(*W*) or O(log*N*). BSOL algorithm [3] is a very attractive algorithm, which only needs at most 7 hash probes in IPv6. Unfortunately, it is a static scheme which makes it unsuitable for dynamic routing table in reality.

For the rest few IPv6 specific algorithms, most of them do not support incremental update [10][11] as they include too many compact techniques or prefix reputation. In one of our pervious work [12], we introduced a range-based dynamic IPv6 lookup scheme BTLPT (balance tree with LPFST) with O(log*N*) performance, which excels all existing IPv6 dynamic algorithms.

In conclusion, the performance of existing dynamic schemes can not be better than O(*W*) or O(log*N*). As a result, their performance will either descend with longer address length in IPv6 or with larger routing table in the future.

III. A DIVISION TECHNIQUE FOR ROUTING TABLE

A. division related definition

- disjoint, overlap

Suppose there are two ranges: a∈[a1,a2](a1≤a2), b∈[b1,b2](b1≤b2), we call these two ranges are:

disjoint: if $a1>b2 \vee a2<b1$

overlap: if $(a1\geq b1 \wedge a2\leq b2)\vee(b1\geq a1 \wedge b2\leq a2)$

In routing lookup, a prefix can be expressed by two end points: start point and end point. These prefix ranges can

No	Prefix	Next Hop
A	00*	N1
B	01*	N2
C	11*	N3
D	0100*	N4
E	0101*	N5
F	1011*	N6
G	010101*	N7

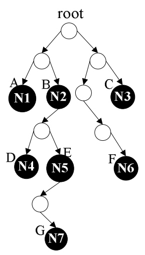

Figure 1. Example of a binary search trie

be only in these two situations: disjoint and overlap. When arranged in one-bit trie (dark nodes in Figure1), the ones on different branches are disjoint (e.g. A and B) while the ones with parent-child (e.g. B and G) relationship are overlapped with each other.

- most specific range, more/less specific range

These definitions are to express prefix relationship from range aspect. Suppose there is a set of prefixes $S=\{A_1, A_2, ..., A_n\}$ arranged in one-bit trie.

most specific range: If A_i is the longest prefix in one branch, we call A_i is a most specific range.

more/less specific range: If A_i can overlap A_j, we call A_i is a less specific range than A_j and A_j is a more specific range than A_i.

- prefix-range set, disjoint set, overlap set

prefix-range set (PRS): Given a set of ranges, if any range can be expressed in a prefix format, we call it a PRS. Routing table is naturedly a PRS.

disjoint set (DS): Given a PRS, if any two ranges are disjoint, we call it a DS. In one-bit trie, the prefixes on different branches can be constructed to a DS, e.g.{A, E, F} is a DS in Figure 2. A routing table can have different DSs.

overlap set (OS): Given two PRSs, say *A and B* $(A \subseteq B)$, if any range in *A* can be overlapped by at least one other range in *A* and no other range in (*B*-*A*) can overlap any range in *A*, we call *A* an OS. In one-bit trie, prefixes on the same subtree can be constructed to an OS, e.g.{B, D, E, G} is an OS in Figure 1. Similarly, a routing table can have different OSs.

- maximum disjoint set

maximum disjoint set (MDS): MDS is the DS with maximum number for a given routing table.

Here, we propose a greedy algorithm to find a MDS for a given routing table (in Figure 2). Suppose *T* is the routing table and *M* is the selected MDS.

There are still several different MDSs for a given routing table and the difference lies on the function of find_left(). In this function, if we encounter several prefixes with the same smallest left end point, we should select one prefix to be r according to some criteria, e.g. select the most specific one.

```
find_MDS(T, R , M) //R is a temp variable
{R = T; M = NIL; //initialize R and M
 sort(R_left, R) //sort the range in R with left end point
  while(R_left != NIL)
  { r = find_left(R_left); //find the smallest left end point
    add(M, r);//add r to M
    remove(R_left, r); //remove r from R_left
   remove(R_left, r_OL);//r_OL prefixes overlap with r}
```

Figure 2. The algorithm to find MDS

TABLE I. MDS OCCUPATION IN IPV4/IPV6 ROUTING TABLE

	MDS	*TOTAL*	*percentage*
Japan(IPv6)	587	606	96.39%
Va(IPv6)	624	665	93.83%
London(IPv6)	622	660	94.24%
Usa(IPv6)	627	665	94.28%
Tilab(IPv6)	612	652	93.86%
Potaroo(IPv6)	612	682	89.88%
as4637(IPv4)	145093	158785	91.47%
as1221(IPv4)	146680	160434	91.42%
as6447(IPv4)	168557	184252	91.48%
V6Gen(100k)	92246	100000	92.24%
V6Gen(200k)	186264	200000	93.13%

- Characteristic of MDS

To obtain the characteristic of MDS in IPv6, we calculate the occupation of MDS for IPv6 real world routing tables. Since IPv6 is still in its initiation period, real world IPv6 routing tables are too small to depict the characteristic, we further use two kinds of routing table to infer the feature: 1) current large IPv4 routing table, which can reflect the influence of hierarchy address allocation policy shared by IPv4 and IPv6, 2) synthetically routing table V6Gen[13], which was developed to forecast future IPv6 routing table. These six IPv6 backbone BGP routing tables are collected on July 3, 2006. Four of them are from four different peers (Vatican, United Kingdom, USA, and Japan) in the RouteViews project[1] and the other two are from Tilab [2] and Potaroo [3] respectively. These three IPv4 backbone BGP routing tables are collected on 2006, Jan. 02 with AS number of as4637, as1221 and as6447. From the result, we can see that both in current IPv4 and IPv6 routing table, MDS occupy around 90 percent of total prefixes (Table 1). The same results can also be got from V6Gen[13] with IPv6 large routing table. **So we can have the conclusion that both IPv4 and IPv6 routing tables are disjoint-prevalent.**

B. *Division technique*

Normally, a routing table can be divided into different DSs. For a DS, since there is no more overlap prefixes, only at most one single match will meet in each DS. This is an exact matching problem which can not only simplify searching process and reduce memory cost, but also support incremental update easily. However, when there are multiple DSs in the data structure, it is much more difficult to deal with the relationship between different DSs. So schemes with multiple DSs will often have dynamic problem.

While for an OS, a prefix can find out all possible matching prefixes in a single OS and thus reduce the total searching space into a much smaller one. It will be suitable for TCAM if the number of entries in the OS is limited.

So in our division solution, we use only one DS together with the result OS. Thus, the advantage of DS and OS can be combined and it can also get rid of updating problem. Our Division Technique is as follows:

A IPv4/IPv6 routing table can be divided into two parts: one for a single MDS and the other for OS.

[1] http://www.routeviews.org/

[2] http://net-stats.ipv6.tilab.com/bgp/bgp-table-snapshot.txt

[3] http://bgp.potaroo.net/v6/as1221/bgptable.txt

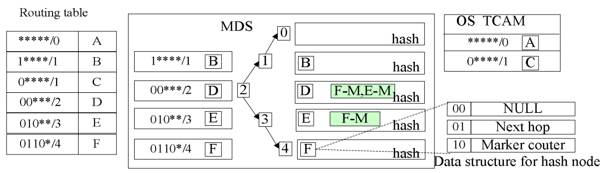

Figure 3. Overall data structure for MDS-OS

IV. PROPOSED SCHEME: DBH

A. *Overall data structure*

With our division technique, the routing table has been split into two parts: MDS and OS. As routing lookup is a longest prefix matching problem, we further use MDS with most specific range as an example. In our example, the range in the MDS is the most specific one among all prefixes in its possible matching list. Thus, searching process can stop immediately when meeting a match in MDS. For example, for the routing table in Figure3, MDS with most specific range is B, D, E, F and the rest OS is A and C.

In the following, we construct different data structures for the above two parts individually. For MDS we select hash tables to perform binary search, which can either implemented in hardware hash unit or in memory. For the rest OS, as the number is limited, they are suitable to fit in TCAM to accomplish O(1) lookup without consuming too much power and memory. Furthermore, we can adopt updateable TCAM structure as in [4]. The overall data structure is shown in Figure 3.

In MDS structure, we organize prefixes with the same length in each hash table and there are O(W) distinct hash tables in total. In order to accomplish binary search on these hash tables, we first construct the overall binary searching path as depicted in Figure3. We should notice that each prefix has its own searching path, which is determined by the prefix length. For example, the searching path for prefix with length 1 is 2-1.

Next, we add markers for each prefix on its own searching path to indicate there might be matching prefixes with longer length. For example, the prefixes with length 4 should label markers on length 2 and 3 in the searching path of Figure3. In the same way, E should label marker on length 2. Thus, the result hash table of length 2 contains D, together with marker F and marker E. We should notice that once the searching path has determined, the number of markers for each prefix is also determined and every prefix will have at most O(log$W$) markers.

For each hash node, the first two bits indicate the type of this node: 00 for null, 01 for prefix and 10 for marker. For a prefix node, the next hop field is its corresponding interface. While for a marker node, the marker counter sums up the total number of prefixes generating the same marker in this node and it will be used when updating. In our experiment, we choose marker counter as 14 bit long, which can count up to 16384 prefixes generating same marker on one position. It is compeletly adequent for large routing table. In addition, although there are different prefixes generating markers on the same position, the marker itself will only occupy 2 bytes for all these prefixes.

```
Function Insert(x) //insert function for MDS-OS
{   stat = search_stat (x, less_specific);
  if(stat ≡ OS) //if it is a OS node
     insert_OS(x); // insert in the OS
  else if((stat ≡ MDS) && (less_specific != NIL)){
//If there is less specific prefix in MDS
     delete_MDS(less_specific); //delete less specific one
     insert_MDS(x); //insert the one to tree
     insert_OS(less_specific)}//insert less_specific one
  else if((stat ≡ MDS) && (less_specific≡ NIL))
      insert_MDS (x);}//if it is a new MDS node
```

Figure 4. Insert procedure of DBH

B. Lookup procedure in DBH

The searching process can be either in serial or in parallel. For serial solution, as the ranges in MDS are most specific ones, we can stop searching when meeting a match in MDS. Otherwise, we continue to search the OS. For example, if there is a prefix of 01010, we first search in MDS hash tables and get a match with prefix E. So, E is result LMP. While for parallel solution, we can also search in MDS and OS simultaneously, and then choose the best matching prefix among two parts. As MDS occupied above 90 percent of total prefixes, most searching procedure can stop in MDS.

C. Insert procedure in DBH

Insert procedure is a little more complicated. If the prefix is to be inserted in OS, there will be no impact on the original ones in MDS and OS. Otherwise, if the prefix is to be inserted in MDS and it is a more specific range than the original one in MDS, we should firstly determine whether there is a less specific range in MDS than the inserting one and which one is it. Then, we should remove the original prefix from MDS and insert it in OS in case the inserting prefix is a more specific range than the original one. As the prefixes in MDS are disjoint, there will be at most one prefix in MDS whose range is less specific than the inserting one. So prefix to be changed can be limited to the specific one.

For example, if there are three prefixes, i.e., A, B, C in our routing table (as shown in Figure 3), where {B, C} is a MDS currently. If we want to insert D in this routing table, whose range is more specific than C, we have to delete C from MDS and insert it in OS.

The pseudocode of Insert(x, length) is as Figure 4. In this process, search_stat function is to find out a less specific range in MDS than the inserting prefix. In function of delete_MDS and insert_MDS, some changes should be conducted on corresponding markers of searching path, by either changing the marker counter or changing the node.

```
Function Delete(x)
{stat = search_stat(x, delete_node);
      if(delete_node ≡ NIL)
         delete(x, stat); //delete x either from MDS or OS
      else if(delete_node != NIL ){
         delete_OS(delete_node);//delete it from OS
         insert_MDS(delete_node);//insert it into MDS
         delete_MDS(x); //delete x from MDS}}
```

Figure 5. Delete procedure of DBH

TABLE Ⅱ MDS PREFIX LENGTH DISTRIBUTIONS OF THE IPV6 ROUTING

	Tilab	**Potaroo**	**VA**	**UK**	**USA**	**Japan**
16	0	1	0	0	1	0
19	1	0	1	0	0	0
20	2	1	2	2	1	1
21	2	0	1	1	1	2
24	1	1	0	0	4	4
28	2	1	2	1	3	4
30	1	1	1	1	1	1
32	**501**	**503**	**501**	**498**	**502**	**486**
33	5	5	5	5	5	5
34	5	5	5	5	5	5
35	**31**	**33**	**31**	**31**	**31**	**31**
40	3	15	8	6	8	3
42	0	10	1	1	1	0
48	**48**	**76**	**66**	**71**	**66**	**18**
64	6	0	6	2	0	5
128	0	0	0	0	0	1

type. As there is at most one less specific needs to be changed in MDS or OS, and the number of corresponding markers to be modified in MDS is O(log*W*), the overall performance for update is good.

D. Delete procedure in DBH

Deleting procedure is similar with insert procedure. It has to determine whether there is a prefix in OS, which becomes a prefix of MDS after deleting some prefix in MDS. For example, the original routing table is {A, B, C, D} in Figure3, {A, C} for OS and {B, D} for MDS. After deleting prefix D, prefix C becomes a prefix of MDS. Hence, it should be deleted from OS and than inserted in MDS.

The pseudocode of *delete(x)* is as Figure.5. There will be at most one prefix to be affected between MDS and OS.

E. Optimized Searching path for IPv6 routing table

In reality, MDS prefix distribution along different length differs dramatically and the hierarchy allocation policy in IPv6 will also make it true in the future. As a result, we can make use of these characteristic to choose more efficient searching path. The prefix length distributions for MDS of the above 6 IPv6 routing tables are listed in TableⅡ.

We can see from the table that: 1) most of the MDS prefix lengths are less than 64 bits; 2) there is no prefix with prefix length less than 16; 3) the routes peak at lengths of 32, 35, and 48, the bolded rows in the table; 4) the total number of unique MDS prefix lengths in the combined distribution is 16, greatly less than the length of IPv6 address 128.

For future IPv6 consideration, according to RFC 3177 (IAB/IESG recommendation on IPv6 address allocation to sites) and RIPE 267, the IPv6 address blocks should be allocated to subscribes following these rules: 1) '/48' in the general case, except for very large subscribes, which could receive a '/47' or multiple '/48s'; 2) '/64' when it is known that one and only one subnet is needed by design; 3) '/128' when it is ABSOLUTELY known that one an donly one device is connecting. So the future IPv6 routes should peak at lengths of 48, 47, 64, and 32.

With this discussion, we can choose the searching path in Figure6. The left part is the searching path for current IPv6 routing table and the right part is for future routing table

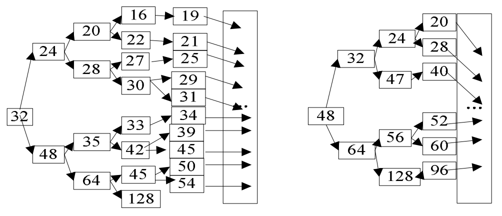

Figure 6. Optimized searching path of DBH

V. PERFORMANCE EVALUATION

We implement the above data structure and algorithm with standard C, and simulate the actual process of IPv6 router lookup under Linux on an Intel Pentium4 PC with CPU of 2.4GHz and 512M DDR333 memory, and Red Hat Enterprise Linux Advanced Server 4.0.

To measure the average lookup memory access number, we generate one million IP addresses for each test run. The verification IP addresses are not completely random ones, 80% of which are generated based on a routing entry randomly selected from the corresponding routing table and the other 20% are generated purely randomly. Here, we also use two types of evaluations in our experiment: 1) Using the real world IPv6 backbone BGP routing tables obtained world wide; 2) Using the synthetically created IPv6 routing tables to test the scalability of our scheme generated by V6Gen[13], to simulate large routing tables.

Figure7 (left) is the lookup comparison result for current IPv6 routing table. The performance of DBH with optimized searching path are compared with those of LPFST[7], LC[6], Patricia[5] and BTLPT[12], the latest IPv6 dynamic LPM algorithm. We can see that the average lookup memory access number is only 2.7 for current IPv6 routing tables, which is only 35% of BTLPT, 13% of LC and much better than other schemes. Figure7 (right) is the scalability comparison result for future IPv6 routing table. We can see that our DBH scheme can have a quite stable performance since its performance has nothing to do with the size of routing table. As a result, it will become even more attractive in the future IPv6 compared to other schemes.

Memory consumption and average update access number compared with other schemes, are presented in TableⅢ. As many existing algorithms do not support incremental update, we can only compare it with BTLPT and LPFST. Here, we choose two routing tables for example. In DBH hash structure, there is no need to store prefixes any more, so its memory cost is extremely small. Actually, every hash function may have some memory waste and we have to

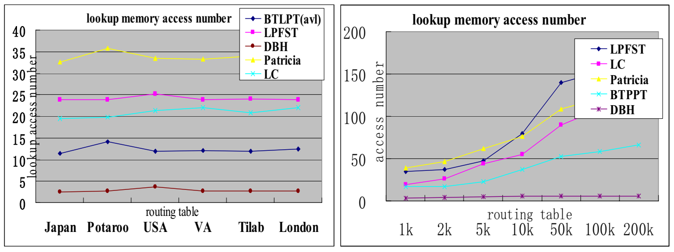

Figure 7. Lookup memory access number comparison

TABLE Ⅲ MEMORY AND UPDATE COMPARISON

Right: total number	Japan(total:587)	London(total:625)
Mem:DBH	4.28 (KB)	5.65 (KB)
Mem:BTLPT	28.68 (KB)	31.06 (KB)
Mem: LC	35.64 (KB)	39.19 (KB)
Mem: Patricia	33.20 (KB)	33.20 (KB)
Mem: LPFST	36.48 (KB)	35.78 (KB)
Update: DBH	23.11	22.99
Update:BTLPT	105	24.29
Update: LPFST	2.81	69.28

choose a good hash function which is beyond the scope of this paper. Here, we just emphasis that the memory consumption cost by this data structure itself is extremely small. For update comparison, we use half of the real world routing table to construct initial data structure and the rest prefixes to be inserting ones. The result in TableⅢ also shows that our scheme only needs around 23 memory accesses for dynamic update per-prefix.

VI. CONCLUSION

In this paper, we propose a novel division technique for routing table. Furthermore, we introduce our dynamic binary hash scheme based on this division technique. The performance evaluation has shown that DBH has much better performance than all existing dynamic schemes.

REFERENCES

[1] Lu, H. and S. Sahni., "$O(log W) Multidimensional Packet Classification.", Networking, IEEE/ACM Transactions on, pp. 462-472, April. 2007

[2] Pi-Chung, W., L. Chun-Liang., "Performance Improvement of Two-Dimensional Packet Classification by Filter Rephrasing", IEEE/ACM Transactions on Networking, pp. 906-917, Aug. 2007

[3] Waldvogel, G. Varghese, J. Turner and B.A. Plattner "Scalable High Speed IP Routing Lookups", ACM SIGCOMM'97, pp.25-36, Sept.1997

[4] Haoyu, S; Jonathan, T, "Fast Filter Updates for Packet Classification using TCAM", GLOBECOM '06, pp 1-5, Nov 2006

[5] D.R. Morrison, "PATRICIA – Practical Algorithm to Retrieve Information Coded in Alphanumeric" J.ACM, Vol.15, pp.514-534, Oct. 1968.

[6] S. Nilsson and G. Karlsson, "IP-Address Lookup Using LC-Trie" IEEE J. on Sel. Area in Comm, Vol.17, pp.1083-1092, June.2001

[7] L.C. Wnn, K.M. Chen and T.J. Liu, "A Longest Prefix First Search Tree for IP Lookup" in ICC'05, pp..989 – 993, May.16-20, 2005

[8] P.R.Warkhede, S.Suri, and G.Varghese, "Multiway Range Trees: Scalable IP Lookup with Fast Updates", Computer Networks, vol.44, no.3,pp.289-303, 2002

[9] H.Lu,S.Sahni, "A B-Tree Dynamic Router-Table Design", IEEE Trans.Computers,vol.54,pp.813-823,2005

[10] Kai Zheng, Chengchen Hu Zhen Liu, Bin Liu, "A Scalable IPv6 Route Lookup Scheme via Dynamic Variable-Stride Bitmap Compression and Path Compression," in Computer Communications, pp 3037-3050, 2006

[11] Z. Li, X. Deng, H. Ma, and Y. Ma, "Divide-and-conquer: a scheme for IPv6 address longest prefix matching," in IEEE Workshop on High Performance Switching and Routing, pp. 37-42, Jun. 2006.

[12] Q.Sun, X.Y.Zhao, H.X.Hong, W.J.Jiang and M. Yan, "A Scalable Exact matching in Balance Tree Scheme for IPv6 Lookup" ACM SIGCOMM IPv6'07 ,pp 397~402, August 31, 2007

[13] K. Zheng and B. Liu, "V6Gene: A scalable IPv6 prefix generator for route lookup algorithm benchmark," in IEEE AINA, Vienna, Austria, pp. 147-152, Apr. 18-20, 2006.