An Efficient IP Address Lookup Algorithm Using a Priority Trie

Hyesook Lim
Information Electronics Engineering
Ewha Womans University
Seoul, Korea
hlim@ewha.ac.kr

Ju Hyoung Mun
Information Electronics Engineering
Ewha Womans University
Seoul, Korea

***Abstract*—Fast IP address lookup in the Internet routers is essential to achieve packet forwarding in wire-speed. The longest prefix matching for the IP address lookup is more complex than exact matching because of its dual dimensions, length and value. By thoroughly studying the current proposals for the IP address lookup problem, we find out that binary search could be a low-cost solution while providing high performance. Most of the existing binary search algorithms based on trie have simple data structures which can be easily implemented, but they have empty internal nodes. Binary search algorithms based on prefix values do not have empty nodes, but they either construct unbalanced trees or create extra nodes. In this paper, a new IP address lookup algorithm using a priority trie is proposed. The proposed algorithm is based on the trie structure, but empty internal nodes are replaced by priority prefixes. The longest prefix matching in the proposed algorithm is more efficiently performed since search can be immediately finished when input is matched to a priority prefix. The performance evaluation results show that the proposed priority trie has very good performance in terms of the memory requirement, the lookup speed, and the scalability.**

Keywords-Internet protocol; Address lookup; Binary trie; Priority trie

I. INTRODUCTION

The rapid growth of the Internet traffic requires for routers to perform high-speed packet forwarding. As an essential component to achieve fast packet forwarding, the Internet protocol (IP) address lookup is one of the most challenging tasks to the Internet routers since it should be performed in wire-speed for each incoming packet under the circumstance that packets' arrival rates and the size of routing tables have been dramatically increased [1]. The IP address lookup is to determine the output port using the destination IP address of incoming packets in order to forward the packets toward their final destinations.

IP addresses have two levels of hierarchy: network part and host part. The network part is called prefix. Class-less inter-domain routing (CIDR) structure allows the arbitrary length of prefixes and address aggregation at multiple levels. As a result, the IP address lookup problem in routers requires searching the forwarding table for the longest prefix that matches the destination address of a given input packet in order to find the most specific route. Determining the longest matching prefix (LMP) or the best matching prefix (BMP) for the IP address lookup now involves two dimensions: length and value [2].

High-performance routers facilitates hardware parallelism using specialized memories called ternary content addressable memory (TCAM), in which an address lookup is performed with a single memory access. However, as a power-hungry device, TCAM is a lot more expensive than ordinary memory in implementation cost. A lot of algorithms and architectures performing the longest prefix match using ordinary memories have been proposed.

A set of metrics is used in evaluating the performance of the IP address lookup algorithms. Search speed is the primary metric and it is highly dependent on the number of memory accesses for table lookups since memory access is the most time consuming operation in the search process [3]. The size of the required memory is also an important metric according to the growth of the size of routing tables to several hundred thousand entries. For routing tables which have dynamically changed routes, providing incremental updates is also important. Scalability is another important metric, in which algorithms should be easily modified for accommodating large routing tables.

In this paper, a new IP address lookup algorithm using a priority trie is proposed. Most of the trie-based algorithms have many empty internal nodes which cause excessive memory space and memory accesses. Moreover, the trie-based algorithms compare a given input with shorter prefixes first, and hence search has to be continued until a leaf is visited even if a match is found. The proposed algorithm removes empty internal nodes replacing by the longest prefix among the prefixes belonged to the sub-tree of each empty node as a root. Hence the memory requirement is reduced. The replaced node is called priority node. Search in the proposed algorithm is immediately finished without searching the complete trie if a given input matches a priority node, and hence search performance is significantly improved.

This paper is organized as follows. In Section II, we briefly summarize the related works for IP address lookup. Section III

This research was partially supported by the Ministry of Information and Communication, Korea, under the HNRC-ITRC support program supervised by the Institute of Information Technology Assessment.

1-4244-0357-X/06/$20.00 ©2006 IEEE

presents our proposed algorithm. In Section IV, extensive simulation results including the comparison with the related works are shown. Section V concludes the paper.

II. Related Works

A. Algorithms based on hashing

Hashing has been popularly used for layer 2 address lookup which requires exact matching. Hashing converts a long length string into a smaller length which can be used as a memory pointer, and hence collision is the intrinsic problem of hashing. Broder et al. proposed to use multiple hash functions in order to reduce collisions [4]. For the IP address lookup, hashing is applied into prefixes of the same length, and the longest prefix among matched prefixes in each length is selected as the best match [5]-[7]. Waldvogel et al. proposed to use binary search on hash tables organized by prefix lengths [5]. Lim et al. proposed to use multiple hash functions in reducing collisions and perform parallel search for every hash tables in each length [6]. Other interesting approach is to combine the hashing and the binary search [7]. In their approach, hashing is primarily applied into prefixes of the same length, and for prefixes collided into the same entry, binary search is performed.

B. Binary search algorithms based on trie

A trie is the most intuitive data structure for the IP address lookup [2][8]-[10]. The trie is a tree-based data structure which applies linear search on length. Each prefix resides in a node of the trie, of which the level and the path from the root node is uniquely determined by the length and the value of the prefix, respectively. Figure 1 shows the binary trie for an example set of prefixes. In Figure 1, black nodes represent prefixes, and white nodes represent unassigned internal nodes. At each node, search in the binary trie proceeds to the left or right according to the sequential inspection of address bits starting from the most significant bit.

The binary trie is a natural way to represent prefixes, but it is not balanced and the depth of the trie is usually W, where W is the maximum prefix length. Moreover, because of unassigned internal nodes included in the trie, memory space is wasted. Another intrinsic problem of the trie is that shorter prefixes are located in a higher level and hence they are compared earlier than longer prefixes. Therefore, even if a match is found, search has to be continued until a leaf is visited in order to look for a longer match.

In order to reduce the depth of the trie, multi-bit trie inspects more than one bit at a time [8], and path-compressed trie collapses one-way branch nodes [2]. Level-compressed trie applies multi-bit trie with path compression [9]. In order to save memory by compression, Lulea algorithm proposed a compact trie structure for fast lookup [10], but it requires a lot of pre-processing and hence does not allow incremental updates.

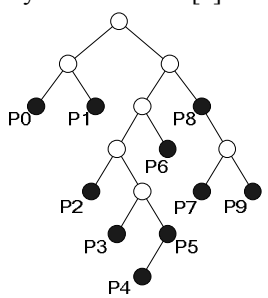

Figure 1. The binary trie for an example set of prefixes

C. Binary search algorithms based on prefix value

Binary prefix tree (BPT) [11] algorithm attempts to perform binary search on prefix values. In order to perform the binary search on prefix values, prefixes should be sorted according to their magnitude. The BPT scheme provides a set of new definitions for the comparison of prefixes of different lengths in sorting prefixes in the order of magnitude. For two prefixes of different lengths, the first m bits are compared, where m is the length of the shorter prefix. The prefix having the bigger value is defined as a bigger prefix. If they are the same, then the $(m+1)^{th}$ bit of the longer prefix is checked. If the $(m+1)^{th}$ bit is 1, the longer prefix is bigger, and otherwise the shorter prefix is bigger.

However, the binary search can not be directly applied to this sorted list because of prefix nesting relationship. Assuming that an incoming packet which has the prefix A as a best matching prefix is given, it is possible to exist prefixes which have the prefix A as its sub-string. If such prefixes are compared earlier than the prefix A with the input, the binary search can be directed to the wrong half of the list which does not include the prefix A. The BPT algorithm solves this issue by restricting ancestor prefixes being compared earlier than descendant prefixes. The BPT does not have empty internal nodes, and hence it has the advantage in required memory size. However, depending on the depth of the prefix hierarchy, tree depth could become very large as will be shown in Section IV.

As an attempt to reduce the depth of tree, the weighted prefix tree (WPT) [12] considers the number of descendents in selecting the root of each level. The constructed WBPT has a shorter depth and is more balanced than BPT. Using the fact that disjoint prefixes construct a perfectly balanced tree, the multiple balanced prefix trees (MBPT) [13] constructs multiple balanced trees only with disjoint prefixes. The disjoint prefix tree (DPT) [14] constructs the BPT for leaf-pushed prefixes, and hence it is a perfectly balanced tree for the extended set of prefixes generated by the leaf-pushing. The binary search on range (BSR) [15] treats each prefix as an interval which has a start address and an end address. The start and the end addresses are defined by padding zeros and ones to the maximum length, respectively. Hence the length dimension is completely removed. However, since the start and the end addresses are stored into the routing table, the number of elements could be the twice of the actual number of prefixes in the worst case. For each disjoint interval, the BSR scheme has to pre-compute and store the BMP, and the binary search is performed on the list of intervals. Incremental update is not possible because of the pre-computation of BMPs for each interval.

III. Proposed Algorithms

As mentioned earlier, not only the binary trie has many empty internal nodes but also shorter prefixes are stored at

1-4244-0357-X/06/$20.00 ©2006 IEEE

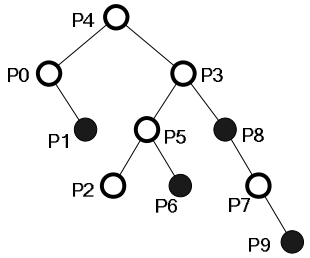

Figure 2. The proposed priority-trie

higher levels and hence they are compared earlier than longer prefixes. Therefore, even if a match is found, search has to be continued until a leaf is visited. If prefixes are reversely assigned, in other words, if longer prefixes are associated with higher level nodes and shorter prefixes are associated with lower level nodes, search can be finished immediately when there is a match. However, in order for a prefix to be associated with a node in a lower level than its length, the prefix has to be duplicated 2^{j-i} times, where i is the length of the prefix and j is the located level.

In this paper, we propose to use the empty internal nodes in locating longer prefixes at higher levels. In other words, if we associate an empty node with the longest prefix among the prefixes belonged to a sub-tree rooted by the empty node, empty nodes are completely removed, the depth of a trie would be reduced, and search would be more efficient. The prefixes associated with empty nodes are named priority prefixes. Since we only associate empty nodes with longer prefixes and prefixes are located in either the same level or the higher level than its prefix length, there is no prefix duplication in our algorithm.

Figure 2 shows the proposed priority-trie using the same example set of prefixes as in Figure 1. The black nodes represent the prefixes located in their own levels, and the nodes with bold boundary represent the priority prefixes. In Figure 2, since the prefix P4 is the longest prefix belonged to the binary trie of the empty root, it is located into the root node. There are two prefixes with the same length belonged to the sub-tree of the empty node 0*. Here we assume to break the tie from the left prefix, and hence the prefix P0 becomes the priority prefix. The prefix P3 is the longest prefix among the prefixes belonged to the sub-tree of the empty node 1*, and hence the prefix P3 is located into 1*, and so on.

A. Building the proposed priority-trie

Building the proposed priority-trie is composed of following four steps. In the first step, prefixes are listed in the increasing order of their lengths. Since the optimum depth of the binary trie is $\lceil \log_2(N+1) - 1 \rceil$ for N prefixes, in the second step, prefixes with the length less than or equal to L are stored into the corresponding level of nodes in the binary trie, where $L = \lceil \log_2(N+1) - 1 \rceil$. In the third step, starting from the longest prefix in the list, follow the search path and store the prefix into the first empty node met in the path or create a leaf and mark the prefix as a priority prefix. Repeat this step until every prefixes longer than L are located. In the final refinement step, if the number of nodes is greater than N, this means that empty nodes still exist. Starting from the bottom-left prefixes to the bottom-right prefixes, remove the prefix, follow the trie, and locate it into the first empty node met in the path and mark the prefix as a priority prefix. Repeat this step until the number of nodes is equal to N.

TABLE I. ROUTING TABLE FOR THE PROPOSED PRIORITY TRIE

addr	Priority /ordinary	prefix	length	leftPtr	rightPtr	Out port
0	1	100110*	6	1	2	P4
1	1	00*	2	-	3	P0
2	1	10010*	5	4	5	P3
3	0	01*	2	-	-	P1
4	1	10011*	5	6	7	P5
5	0	11*	2	-	8	P8
6	1	1000*	4	-	-	P2
7	0	101*	3	-	-	P6
8	1	1110*	4	-	9	P7
9	0	1111*	4	-	-	P9

Table I shows the routing table built by the proposed algorithm. The first column is the memory address and it can be arbitrary. The second column represents whether the stored prefix is a priority prefix (1) or an ordinary prefix (0). Prefixes and their lengths are stored in the third and the fourth column, respectively, and two child pointers are shown in the following columns. The last column is the output port corresponding to the prefix, and we put the prefix name in this column for simplicity. As shown, the number of routing entries in the proposed algorithm is equal to N.

B. Search

The search procedure in the proposed priority trie is shown in Figure 3. Same as the search in the binary trie, search proceeds to the left or right according to the sequential inspection of address bits starting from the most significant bit. However, search in the proposed algorithm is finished either at a match with a priority prefix or at a leaf while it is always finished at a leaf in the binary trie.

C. Update

For the deletion of a prefix in the proposed priority trie, the prefix is located and the first three fields of the located entry are deleted. Since this node still contains the child pointers so

```
Search (input)
BMP = *;
ptr = index(root); //start at root
do {
      if (input == prefix(ptr))  // input matches to a prefix
      {
            BMP = prefix(ptr);
            if (priority(ptr) == '1')
                  break; //matched prefix is a priority prefix
      }
      if (nextBit(input) == '0')
            ptr = leftPtr(ptr); // follow the left pointer
      else    ptr = rightPtr(ptr); //follow the right pointer;
} while (ptr ! = NULL)
return BMP;
```

Figure 3. Search procedure in the proposed priority-trie

1-4244-0357-X/06/$20.00 ©2006 IEEE
This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the IEEE GLOBECOM 2006 proceedings.

```
Update (inputPrefix)
curLevel = 0;
newPfx = inputPrefix;
ptr = index(root); //start at the root
do {
    if (newPfx == prefix(ptr)) // a match
    {
       if (priority(ptr) == '1') //match a priority prefix
            if ( (length(newPfx) > length(ptr))
                 or (length(newPfx) == curLevel)  )
                 {
                       tmp = prefix(ptr);
                       store newPfx into the ptr;
                        if (length(newPfx) > curLevel)
                             priority(ptr) = 1;
                       else  priority(ptr) = 0;
                       newPfx = tmp;
                 }
           else if (length(newPfx) == length(ptr))
                store newPfx into the ptr;
                break;
      else //match an ordinary prefix
            if (length(newPfx) == length(ptr))
                 store newPfx into the ptr;
                 break;
    }
    if (nextBit(newPfx) == '0')
        ptr = leftPtr(ptr);
    else ptr = rightPtr(ptr);
    curLevel++;
    if (ptr == NULL)
        create a ptr;
        store newPfx into the ptr;
        priority(ptr) =1;
} while (ptr ! = NULL)
```

Figure 4. Update procedure in the proposed priority trie

that the search process continues to lower levels, we should maintain the node. If we assume that the entire routing table is rebuilt with appropriate regular intervals, the number of empty nodes by the prefix deletion would not be a problem.

For the insertion of a new prefix in the proposed algorithm, we have shown the update procedure in Figure 4. There are two cases that multiple nodes are affected by an insertion. The first case is when the new prefix matches a priority prefix and it is longer than the priority prefix, and the second case is that the ordinary node of the new prefix was taken by other priority prefix. In these cases, the new prefix takes the place, and for the priority prefix, in which its place is taken away, the same procedure is repeated. According to the reported statistics [3], the number of nested networks is less than 7, and hence the number of nodes affected by a prefix insertion would be statistically limited by 7. All other cases, either the new prefix is stored at a leaf or the prefix in the routing table is replaced by the new prefix.

TABLE II. PERFORMANCE OF THE PROPOSED PRIORITY BINARY TRIE

Routing Table	N	N_p	D	D_p	T_a	M (Kbyte)
MAE-West1	14,553	14,199	32	24	16.66	127.9
Aads	20,204	19,568	32	24	17.43	177.6
MAE-West2	29,584	26,671	32	24	18.22	260.0
PORT 80	112,310	50,091	32	28	20.35	987.1
Grouptlcom	170,601	70,525	32	24	20.76	1.46M
Telstra	227,223	119,149	32	32	22.86	1.95M

TABLE III. PERFORMANCE OF THE PROPOSED PRIORITY MULTI-BIT TRIE

Routing Table	N	N_p	N_{extra}	D	D_p	T_a	M (Kbyte)
MAE-West1	14,553	14,388	2,337	16	12	9.70	214.4
Aads	20,204	19,124	3,208	16	12	10.05	297.2
MAE-West2	29,584	23,016	6,273	16	13	10.54	455.2
PORT 80	112,310	36,362	21,630	16	16	11.27	1.79M
Grouptlcom	170,601	47,073	32,732	16	13	11.45	2.71M
Telstra	227,223	75,929	41,850	16	16	12.51	3.85M

IV. PERFORMANCE EVALUATION

Simulations are performed using C language for real routing data from backbone routers [16]. Table II and Table III show performance evaluation results of our proposed priority binary trie and priority multi-bit (2-bit) trie, respectively, in terms of the number of routing prefixes (N), the number of priority prefixes (N_p) among routing prefixes, the maximum prefix length (D), the depth of our proposed priority trie (D_p), the average number of memory accesses (T_a) for an address lookup, and the memory requirement (M) for various sizes of routing data. As shown in the number of priority prefixes in Table II, more than 90 % of the prefixes for the first three routing tables are stored by priorities, and this means that the original binary trie has a lot of empty nodes. The more priority nodes the better search performance is expected in our proposed algorithm. The average number of memory accesses is between 16 and 23, and it is not much degraded as the growth of the routing table size.

In Table III, since 2 bits are considered at the same time in the 2-bit trie, extra nodes generated for fitting into the stride size are shown in N_{extra}. The performance of the proposed algorithm in terms of the trie depth, the average number of

TABLE IV. COMPARISON WITH OTHER ALGORITHMS

Algorithm	incremental update	Port80 (112,310)				Telstra (227,223)			
		T_{max}	T_a	M (MByte)	N_{extra}	T_{max}	T_a	M (MByte)	N_{extra}
Binary trie[2]	yes	32	22.15	1.29	112,907	32	24.64	2.59	225,682
BPT[11]	no	44	25.82	1.25	0	66	30.80	2.60	0
WPT[12]	no	36	20.44	1.25	0	39	23.96	2.60	0
BSR[15]	no	18	11.42	0.96	72,063	19	11.07	1.76	124,795
Proposed binary	yes	28	20.35	0.99	0	32	22.86	1.95	0
Proposed multi-bit	yes	16	11.27	1.79	21,630	16	12.51	3.85	41,850

1-4244-0357-X/06/$20.00 ©2006 IEEE

memory accesses, and the required memory size, is not much degraded as the growth of routing table sizes, and hence the proposed scheme is good in scalability toward large routing data.

We have done performance comparison with existing binary search schemes, one with 112K entries and the other with 227K entries. Table IV shows the performance comparison in terms of the worst case number of memory accesses (T_{max}), the average number of memory accesses (T_a), the required memory size (M), and the number of extra nodes required in the algorithms (N_{extra}). As shown in Table IV, the proposed priority multi-bit trie is the best in the worst-case number of memory accesses. The BSR algorithm and the proposed priority multi-bit trie is the best in the average number of memory accesses. In the required memory size, BSR and the proposed priority binary trie show the best performance. Since BSR algorithm requires the pre-computation of best matching prefixes in each disjoint interval, it does not provide incremental update while the proposed algorithm provides incremental update as described in the previous section.

V. CONCLUSION

As an attempt to remove unnecessary nodes in the trie structure, this paper proposed a new trie-based algorithm for IP address lookup. The proposed algorithm constructs a priority trie, in which each empty node in the trie structure is replaced by a priority prefix which is the longest prefix belonged to the sub-trie rooted by the empty node. Therefore, empty nodes in the trie are completely removed. Search in the proposed priority trie finished either at a leaf or at a match to a priority prefix since it is guaranteed that the matched priority prefix is the longest prefix in the search path. Hence the performance of the proposed algorithm in terms of the memory requirement and search speed is significantly improved compared with the conventional trie structure.

REFERENCES

[1] H. Jonathan Chao, "Next generation routers," *Proceedings of the IEEE*, vol. 90, no. 9, pp. 1518-1558, Sep. 2002

[2] M.A. Ruiz-Sanchex, E.W. Biersack, and W. Dabbous, "Survey and taxonomy of IP address lookup algorithms," *IEEE Network*, pp.8-23, March/April 2001

[3] George Varghese, "Network algorithmics: An interdiciplinary approach to designing fast networked devices," *Morgan Kaufmann Publishers, Elsevier Inc*, 2005

[4] A. Broder and M. Mitzenmacher, "Using multiple hash functions to improve IP lookups", in *Proc. IEEE INFOCOM*, 2001, pp. 1454-1463

[5] M. Waldvogel, G. Varghese, J. Turner, and B. Plattner, "Scalable high speed IP routing lookups," in *Proc. ACM SIGCOMM Conf., Cannes, France*, 1997, pp. 25–35

[6] Hyesook Lim and Yeojin Jung, "A parallel multiple hashing architecture for IP address lookup," in *Proc. IEEE HPSR2004*, Apr. 2004, pp.91-98

[7] Hyesook Lim, Ji-Hyun Seo, and Yeo-Jin Jung, "High speed IP address lookup architecture using hashing," *IEEE Communications Letters*, vol.7, no.10, pp.502-504, Oct. 2003

[8] Sartaj Sahni and Kun Suk Kim, "Efficient construction of multibit tries for IP address lookup," *IEEE/ACM Transactions on Networking*, vol.11, no.4, pp.650-662, Aug. 2003

[9] Stefan Nilsson and Gunnar Karlsson, "IP-address lookup using LC-tries," *IEEE Journal of Selected Areas in communications*, vol.17, no.6, pp.1083-1092, June 1999

[10] M. Degermark, A. Brodnik, S. Carlsson, and S. Pink, "Small forwarding tables for fast routing lookups," in *Proc. ACM SIGCOMM*, 1997, pp.3-14

[11] N. Yazdani and P. S. Min, "Fast and scalable schemes for the IP address lookup problem," in *Proc. IEEE HPSR2000*, pp 83-92

[12] Changhoon Yim, Bomi Lee, and Hyesook Lim, "Efficient binary search for IP address lookup," *IEEE Communications Letters*, vol.9, no.7, pp.652-654, Jul. 2005

[13] Hyesook Lim, Bomi Lee, and Won-Jung Kim, "Binary searches on multiple small trees for IP address lookup," *IEEE Communications Letters*, vol.9, no. 1, pp.75-77, Jan. 2005

[14] Hyesook Lim, Wonjung Kim, and Bomi Lee, "Binary search in a balanced tree for IP address lookup," in *Proc. IEEE HPSR2005*, May 2005, pp. 490-494

[15] B. Lampson, V. Srinivasan, and G. Varghese, "IP lookups using multiway and multicolumn search," *IEEE/ACM Transactions on Networking,* vol.7, no.3, pp 324-334, Jun. 1999

[16] http://www.potaroo.net

1-4244-0357-X/06/$20.00 ©2006 IEEE

This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the IEEE GLOBECOM 2006 proceedings.