LETTER

An Efficient IP Address Lookup Scheme Using Balanced Binary Search with Minimal Entry and Optimal Prefix Vector

Hyuntae PARK[†], **Hyejeong HONG**[†], ***Nonmembers, and*** **Sungho KANG**[†a)], ***Member***

SUMMARY Although IP address lookup schemes using ternary content addressable memory (TCAM) can perform high speed packet forwarding, TCAM is much more expensive than ordinary memory in implementation cost. As a low-cost solution, binary search algorithms such as a binary trie or a binary search tree have been widely studied. This paper proposes an efficient IP address lookup scheme using balanced binary search with minimal entries and optimal prefix vectors. In the previous scheme with prefix vectors, there were numerous pairs of nearly identical entries with duplicated prefix vectors. In our scheme, these overlapping entries are combined, thereby minimizing entries and eliminating the unnecessary prefix vectors. As a result, the small balanced binary search tree can be constructed and used for a software-based address lookup in small-sized routers. The performance evaluation results show that the proposed scheme offers faster lookup speeds along with reduced memory requirements.

key words: *IP address lookup, balanced binary search, minimal entry, optimal prefix vector*

1. Introduction

Due to a tremendous increase in internet traffic and the adoption of optical transmission technologies, routers should be able to forward massive packets at several gigabits per second. For the packet forwarding, IP address lookup determines the output port of incoming packets by looking up a destination IP address in the forwarding table. Nowadays, in order to avoid wasting address space, the prefixes have variable lengths from 8 to 32 bits for the arbitrary aggregation of IP addresses. Therefore, IP address lookup becomes more complicated because the longest matching prefix must be found from among multiple matching prefixes [1].

In order to solve the longest matching prefix problem, high-performance routers have been implemented with hardware parallelism using specialized memories called ternary content addressable memory (TCAM). With TCAM, an address lookup is performed with a single memory access for high-speed packet forwarding [2]. However, TCAM is much more expensive than ordinary memory such as SRAM in circuit complexity as well as power consumption. Accordingly, the current solution for IP address lookup using TCAM is confronted by the cost problems. Therefore, efficient algorithmic solutions are essentially required [3]. A lot of software-based algorithms and schemes performing the longest prefix match using ordinary memories have been suggested. These are implemented in generic or network processors and used for low cost implementation in small-sized access routers. Especially, binary search algorithms such as a binary trie or a binary search tree have been popularly used. This paper has focused on the software-based lookup schemes using a binary search algorithm that can be implemented in small-sized routers.

Several metrics are considered in evaluating the performance of software-based IP address lookup algorithms. The lookup speed is the most crucial metric. Because the cost of computation in the lookup process is dictated by the number of memory accesses, the lookup speed can be evaluated by the average and the worst-case numbers of memory accesses. The size of the required memory is also an important metric according to the growth of forwarding table to several hundred thousand entries. In addition, the incremental updating and the scalability are considered in growing table [1].

The binary trie [1] provides a natural way to find the longest matching prefix, but it requires many empty internal nodes. In the sense that the binary search tree includes no empty internal nodes, it is a better approach than the binary trie. The binary search scheme using a comparison method for sorting values with different lengths has been suggested [4]. However, due to a nesting relationship dependent on a hierarchy of prefixes, there is a limitation that the ancestor (shorter) prefix should be searched earlier than the descendent (longer) prefix. Accordingly, the tree is much unbalanced so that high-speed address lookup is infeasible. As an attempt to eliminate the nesting relationship, a forwarding table consists of multiple trees according to the nesting relationship [5], [6]. In these schemes, because of the hierarchical binary search, the search path via multiple trees is longer than that on a single tree.

On the other hand, a balanced binary search tree is constructed using only prefixes on leaves in a binary trie since these leaf prefixes have no nesting relationship [7]. However, each node should possess a prefix vector containing the nesting information so that the large node size is required. Figure 2 shows the binary search tree with prefix vectors for the example set of prefixes from Fig. 1. In the figure, P_x^o represents the output port o of the leaf prefix P_x, and Y_a^o represents the element of the prefix vector for the output port o of the ancestor prefix P_a. A prefix vector represented as a square bracket contains the forwarding information of the corresponding ancestor prefixes.

In the proposed scheme, a pair of nearly identical entries with duplicated prefix vectors is combined among the

Manuscript received January 19, 2011.

[†]The authors are associated with the Dept. of Electrical and Electronic Eng., Yonsei University, Korea.
a) E-mail: shkang@yonsei.ac.kr

DOI: 10.1587/transcom.E94.B.3128

Prefix		output -port	Prefix		output -port
P_0	000*	1	P_5	11000*	0
P_1	0010*	1	P_6	11001*	1
P_2	0011*	1	P_7	1101*	2
P_3	0*	0	P_8	11*	1
P_4	1*	0	P_9	11111*	2

Fig. 1 Example set of prefixes.

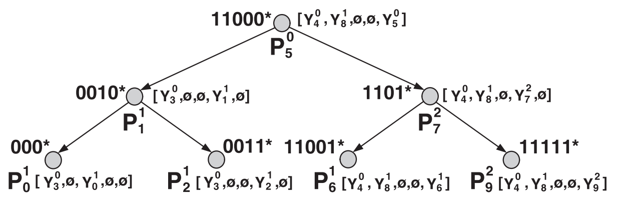
11000* $[Y_4^0, Y_8^1, ø, ø, Y_5^0]$
P_5^0
0010* $[Y_3^0, ø, ø, Y_1^1, ø]$
P_1^1
1101* $[Y_4^0, Y_8^1, ø, Y_7^2, ø]$
P_7^2
000*
P_0^1 $[Y_3^0, ø, Y_0^1, ø, ø]$
0011*
P_2^1 $[Y_3^0, ø, ø, Y_2^1, ø]$
11001*
P_6^1 $[Y_4^0, Y_8^1, ø, ø, Y_6^1]$
11111*
P_9^2 $[Y_4^0, Y_8^1, ø, ø, Y_9^2]$

Fig. 2 Binary search tree with prefix vectors.

entries composed of leaf prefixes in a binary trie. Hence, the number of entries can be minimized and the unnecessary prefix vectors can be eliminated. Accordingly, the proposed scheme can construct a small balanced binary search tree, and then achieve faster lookup speeds along with reduced memory requirements.

2. Proposed Scheme

In order to allow for a balanced binary search, the entries of the proposed scheme consist of leaf prefixes with a prefix vector as in [7]. The depth of a balanced tree is bound to $O(\log N)$, where N is the number of entries. Therefore, the lookup speed of a balanced binary search depends only on the number of entries. In order to improve the lookup speed of a balanced binary search, the number of entries should be minimized. For this purpose, we carefully investigate the leaf prefixes with the prefix vector. As shown in Fig. 2, the prefix strings of P_1^1 and P_2^1 are the same, except for one bit, and their prefix vectors are identical. Likewise, the prefix strings and the prefix vectors of P_5^0 and P_6^1 are similar. Inspired by these facts, a pair of similar entries is combined into a single common entry. In this way, the number of entries can be minimized and the unnecessary prefix vector can be eliminated.

We first define the notation as follows. Let $S(P_x,j)$ be a sub-string of the most significant j bits of prefix P_x. Let $L(P_x)$ and $V(P_x)$ be the length and the prefix vector of prefix P_x, respectively. Let $v(P_x,k)$ be the k-th vector element in $V(P_x)$.

Definition 1 (*Likeness*): For the two prefixes P_A and P_B, P_A is defined as the *likeness* prefix of P_B and vice versa, if and only if $L(P_A) = L(P_B) = l$ and $S(P_A,l{-}1) = S(P_B,l{-}1)$.

Definition 2 (*Iso-likeness*): When the two prefixes P_A and P_B are the likenesses, P_A is defined as the *iso-likeness* prefix of P_B and vice versa, if and only if $V(P_A) = V(P_B)$.

Definition 3 (*Common*): For the two prefixes P_A and P_B, the prefix P_M is defined as the *common* prefix of P_A and P_B, if and only if $L(P_A) = L(P_B) = l$, $L(P_M) = l{-}1$ and $S(P_A,l{-}1) = S(P_B,l{-}1) = S(P_M,l{-}1)$.

A pair of *likeness* or *iso-likeness* prefixes is combined into their *common* prefix. Their prefix vectors are also combined and stored in the prefix vector of the common prefix. In the case of *iso-likeness* prefixes, since their prefix vectors are identical, the prefix vector of the common prefix is equal to the prefix vector of the two prefixes. Otherwise, in the case of *likeness* prefixes, the prefix vector of the common prefix contains different vector elements in conjunction with the vector elements shared by the two prefixes. For the likeness prefixes P_A and P_B, let the last significant one bit of P_A be 0 and that of P_B be 1. Then, assume that P_A and P_B are combined into a common prefix P_M and that the lengths of P_A and P_B are w. According to the last significant one bit of P_A and P_B, the output port of P_A is stored in $v(P_M,w{-}1)$, and the output port of P_B is stored in $v(P_M,w)$. In this case, because the length ($w{-}1$) of the common prefix is always less than the length (w) of the original prefixes by one bit, no additional space is required to store the two output ports. Instead, in order to distinguish between the two cases, the additional *type* field is used. The value at the type filed is either 0 or 1. The type 0 represents the ordinary prefixes and the prefix combined with iso-likeness prefixes. Otherwise, the type 1 represents the prefix combined with likeness prefixes. In the proposed scheme, the entry consists of the type, the prefix string, the length, and the prefix vector. It is represented as $P_x = \{$ type, $S(P_x,L(P_x))$, $L(P_x)$, $V(P_x)$ $\}$.

In Fig. 2, since P_1^1 and P_2^1 are the iso-likeness prefixes, these are combined into their common prefix. This combined prefix and P_0^1 are iso-likeness prefixes and hence these are combined as well. Therefore, the final combined prefix is $P_{M1}^1 = \{ 0, 00^*, 2, [Y_3^0, Y_{M1}^1, \phi, \phi, \phi] \}$. As another example, P_5^0 and P_6^1 are the likeness prefixes which are combined into the common prefix $P_{M2}^{0,1} = \{ 1, 1100^*, 4, [Y_4^0, Y_8^1, \phi, Y_5^0, Y_6^1] \}$.

The building procedure of the proposed search tree involves the following three steps. First, the prefix vector is constructed in each leaf prefix for the corresponding ancestor prefixes. Then, the combination is recursively performed until there are no remaining likeness or iso-likeness prefixes in the leaf prefixes. Finally, the entries are sorted in ascending order. This sorted list becomes both the balanced tree and the forwarding table for a binary search. Figure 3 shows the proposed balanced binary search tree with minimal en-

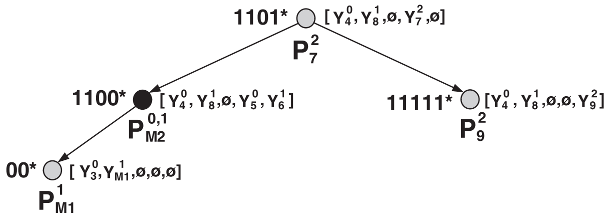
1101* $[Y_4^0, Y_8^1, ø, Y_7^2, ø]$
P_7^2
1100* $[Y_4^0, Y_8^1, ø, Y_5^0, Y_6^1]$
$P_{M2}^{0,1}$
11111* $[Y_4^0, Y_8^1, ø, ø, Y_9^2]$
P_9^2
00* $[Y_3^0, Y_{M1}^1, ø, ø, ø]$
P_{M1}^1

Fig. 3 Proposed balanced binary search tree.

tries for the example set from Fig. 1. The gray nodes represent the prefixes of type 0, and the black node represents the prefix of type 1. As shown, the number of entries can be reduced by more than half of that of the original prefixes in Fig. 1.

The search procedure in the proposed scheme is as follows. Let input A be the destination IP address of the incoming packet. A is defined to exactly match the prefix P_x on node x if $S(A,L(P_x)) = P_x$. In this case, if the prefix is type 0, $v(P_x,L(P_x))$ is the search result. Otherwise, if the type is 1, either $v(P_x,L(P_x))$ or $v(P_x,L(P_x)+1)$ is the search result according to the $(L(P_x)+1)$-th bit in A. Because there is no another matching prefix in the case of an exact match, the search is immediately finished. On the other hand, if $S(A,l_m) = P_x$ when l_m is shorter than $L(P_x)$ and is longer than the length of the previous best matching prefix (BMP), A is defined to partially match the prefix P_x. In this case, the matched prefix string and the corresponding element in the prefix vector of P_x is stored into the BMP, and the binary search continues. If A is smaller than the prefix at the present node, the next node is the medium entry of the smaller half, which is the left child node. Otherwise, the next node is the medium entry of the larger half as the right child node. This process continues until there are no remaining valid nodes. At the end of the tree, the final BMP is set to the search result. As an example of an incoming address, 11001*, an input is initially compared with the root, 1101*. Since the two bits are partially matched, these string 11 and the output port 1 at Y_8^1 is stored into the BMP. Then, the input is compared with 1100*. It matches exactly this prefix of type 1. Thus, the fifth bit of the input is checked. Since this bit is 1, the fifth element of the prefix vector in 1100* is the final search result, regardless of the BMP.

The proposed scheme supports incremental updating. The updating procedure is equal to the method in [7], except in the follow cases. If the new prefix exactly matches the existing prefix of type 0, both prefixes are combined into a new entry of type 1. If the new prefix partially matches the existing prefixes, the output port of the new prefix is stored into the prefix vector of each matched prefix as a new element.

In summary, when N is the number of entries, the search complexity of the proposed scheme is $O(\log N)$, because of the use of a balanced binary search. The updating complexity is $O(\log N)$, as in [7]. The required memory size is determined by multiplying the node size by the number of entries. Thus, the space complexity is $O(N)$.

3. Performance Evaluation

The performance evaluations of the proposed scheme are simulated using C language based on various routing data from real small-sized routers [8]. Table 1 shows the performance comparisons of other binary search schemes in terms of the number of entries in each scheme (N_e), the average (T_{av}) and the maximum (T_{mx}) numbers of memory accesses for the lookup speed, and the required memory size (M).

Table 1 Performance comparisons with other schemes.

Routing data		MaeWest1	Aads	MaeWest2	MaeEast
N		14553	20204	29584	39464
BPT[4]	N_e	14553	20204	29584	39464
	T_{av}	14.1	14.6	15.6	15.8
	T_{mx}	23	22	31	26
	M(KB)	142.1	197.3	288.9	385.4
MBT[5]	N_e	14553	20204	29584	39464
	T_{av}	13.1	13.6	14.6	14.8
	T_{mx}	22	22	26	24
	M(KB)	113.7	157.8	231.1	308.3
SSR[6]	N_e	14533	20204	29407	39455
	T_{av}	13.7	14.4	16.1	16.3
	T_{mx}	39	40	40	40
	M(KB)	167.8	250.4	356.1	478.9
PV[7]	N_e	14288	19832	27622	38316
	T_{av}	12.9	13.4	13.9	14.3
	T_{mx}	14	15	15	16
	M(KB)	348.8	484.2	674.4	935.4
Proposed	N_c	2309	3941	8517	9082
	N_e	12244	16263	21067	30382
	T_{av}	12.7	13.0	13.5	14.0
	T_{mx}	14	14	15	15
	M(KB)	298.9	397.0	514.3	741.7

Table 2 The percent reduction rate as compared with PV.

Data	MaeWest1	Aads	MaeWest2	MaeEast	Ave.
R_T(%)	1.4	2.7	2.7	2.5	**2.3**
R_M(%)	14.3	18.0	23.7	20.7	**19.2**

The numbers of entries (N_e) in BPT, MBT and SSR are less than or equal to the number of original prefixes (N). However, because MBT and SSR consist of multiple trees and the BPT tree is unbalanced, the lookup speeds of these schemes are slow. On the contrary, PV and the proposed scheme use a binary search based on a single balanced tree. Thus, the lookup speeds of both schemes are faster than those of the other schemes. On the other hand, because both schemes should possess a prefix vector containing the nesting information, the memory requirement of both schemes is larger than that of the other schemes.

In the proposed scheme, the number of entries (N_e) is reduced by the cumulative number of performing combination operations in the leaf prefixes (N_c). As shown, the number of entries of the proposed scheme is the smallest. As a result, the lookup speed of the proposed scheme using a small balanced binary search tree is the fastest. Furthermore, because the number of entries is minimized and the wasted prefix vectors are eliminated, the required memory space can be reduced.

Table 2 shows the percent reduction rate of the average memory accesses (R_T) and the required memory size (R_M) as compared with PV. As shown, the average number of memory accesses of the proposed scheme is less than that of PV by 2.3% in average. The memory requirement of the proposed scheme is much smaller than that of PV by 19.2%. Therefore, the memory requirement is considerably reduced, whereas the lookup speed is slightly improved, as compared with PV. Consequently, considering both the lookup speed and the memory requirement, the proposed scheme can be efficiently used for a software-based IP address lookup in real small-sized router.

4. Conclusion

In this paper, we propose a software-based IP address lookup scheme using a new balanced binary search tree with minimal entries and optimal prefix vectors. Among entries composed of leaf prefixes with a prefix vector for constructing the balanced tree, the similar entries are combined. Accordingly, the number of entries can be minimized and the prefix vectors can be optimized. As a result, the proposed scheme provides faster lookup speed with reduced memory requirements for software-based address lookup in small-sized routers.

References

[1] H.J. Chao, "Next generation routers," Proc. IEEE, vol.90, pp.1518–1558, Sept. 2002.
[2] V.C. Ravikumar, R.N. Mahapatra, and L.N. Bhuyan, "EaseCAM: An energy and storage efficient TCAM-based router architecture for IP lookup," IEEE Trans. Comput., vol.54, no.5, pp.521–533, May 2005.
[3] G. Varghese, Network algorithmics: An interdisciplinary approach to designing fast networked devices, Morgan Kaufmann Publishers, Elsevier, 2005.
[4] N. Yazdani and P.S. Min, "Fast and scalable schemes for the IP address lookup problem," Proc. IEEE Int. Conf. on High Performance Switching and Routing, pp.83–92, 2000.
[5] H. Lim, B. Lee, and W. Kim, "Binary searches on multiple small trees for IP address lookup," IEEE Commun. Lett., vol.9, no.1, pp.75–77, Jan. 2005.
[6] H. Park, H. Kim, H.-S. Kim, and S. Kang, "A fast IP address lookup algorithm based on search space reduction," IEICE Trans. Commun., vol.E93-B, no.4, pp.1009–1012, April 2010.
[7] H. Lim, H. Kim, and C. Yim, "IP address lookup for Internet routers using balanced binary search with prefix vector," IEEE Trans. Commun., vol.57, no.3, pp.618–621, March 2009.
[8] http://www.potaroo.net