A New Efficient Binary Search on Range for IP Address Lookup

Hyun-Sik Kim **Hyuntae Park** **Daein Kang** **Sungho Kang**

Department of Electrical and Electronic Engineering
Yonsei University
Seoul, Korea

hskim@soc.yonsei.ac.kr radiob@soc.yonsei.ac.kr abrick@soc.yonsei.ac.kr shkang@yonsei.ac.kr

Abstract – *IP address lookup is an important function to meet such requirements as high speed, memory size and low power for the overall performance of the routers. The biggest problem of IP address lookup is that prefixes has a variable length so that longest prefix match should be employed for IP address lookup. Binary search is useful to decrease search times. But binary search is only adaptable to an exact match. Prefix range search proposed the method binary search can be extended for the longest prefix match. However, this approach requires a lot of memories for the increased set of prefixes and pointers. We focus on boundaries between prefixes which can be removed and the number of pointers. This paper proposes the improved algorithm to reduce the memory requirement in prefix range search dramatically.*

Keywords: IP address lookup, longest prefix matching, binary search, prefix range search.

1 Introduction

As the routing table sizes increase, IP address lookup has become the most critical part of Internet routers because some constraints such as speed and memory sizes are limited. Backbones links have been upgraded to 10Gbps and 40Gbps, and local links upgraded to gigabit speeds [1]. For a 40Gbps line and a 40-byte packet, the processing time of packets at each port must be within 8 nsec (8*40byte / 40Gbps). Therefore, IP address lookup should be done within 8 nsec without a delay in the router because packets come into the router continuously at the wire speed.

In routers, incoming packets and routing tables are stored in memories. The speed of IP address lookup is limited since memory access times are too slow for the wire speed. Since on-chip SRAM is accessible in 0.5 nsec, the number of memory accesses is required to be less than 16. Reducing the number of memory accesses is the key to speed of IP address lookup [2].

Another difficulty in IP address lookup is having to use the longest prefix match of packets rather than an exact match. An exact match is to find the output port associated with the prefix in the routing table which is exactly the same as the packet destination address. Exact matching gives a guaranteed address lookup that can be done at high speeds [3]. However, the Internet uses the prefixes with different lengths which represent hierarchical addressing modes, so that we must use the longest prefix matching to search for a corresponding address.

A lot of algorithms for IP address lookup have been proposed which use the longest prefix matching manner [4][5]. Among these algorithms, some have been studied using binary search which reduces the number of the search for desirable values. Binary search is useful for an exact match, but since prefixes have different lengths, it cannot be applied to IP address lookup directly. Some algorithms show that binary search can be adapted to IP address lookup with longest prefix match [6]. Prefix range search is an example of using binary search to achieve good search time. However, it requires twice the number of entries in the worst case scenario.

This paper focuses on the problem of the prefix range search that uses up to double the number of entries more than the number of prefixes in original routing table. To solve this requirement, we propose a new algorithm.

2 Prefix range search

The basic idea of prefix range searches [3] is using binary search for IP address lookup. It requires several modifications to change the longest prefix match to the exact match for using binary search to search the address. Prefix range search solves this problem by handling IP addresses which have variable lengths.

A prefix can actually be thought of as a range. The prefix has its own space on the dimension of the IP address. Prefix range search makes use of a range

expanded from a prefix. Thus, the endpoint of a range for every prefix is added in the routing table.

To show how to make a table by extending to a range, we used a simple example. Table 1 shows a sample routing table in which there are 4 prefixes. We only expanded the prefixes by the size of 6 to keep simpler instead of 32. Two strings are padded with 0 and 1 from a single prefix for expressing an interval. A prefix 1*(P1) represent a range 100000 to 111111. Hence, one prefix has two points, a starting point and an endpoint, in order to show a range. At this prefix, 100000 is the starting point and 111111 is the endpoint of range for prefix P1.

TABLE I. A sample set of prefixes

	Prefix	Next Hop
P1	1*	H1
P2	101*	H2
P3	101010*	H3
P4	1011*	H4

In this paper, the prefixes padded with 0 and 1 are called *low entry* and *high entr*y, respectively. In addition, we distinguished between *the prefixes* and *the entries*. The prefixes represent all prefixes from the set of prefixes. The entries indicate the some elements in the set of prefixes after expanded and padded with 0 and 1.

After adding the strings padded with 1, the entries are sorted in order of value. The result for four prefixes in table 1 is shown in Figure 1. The lines are connections between the low and high entries made from the same prefix.

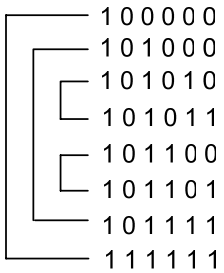

Figure 1. The list of entries padded with 0 and 1.

After making a database which includes 7 ranges, we must re-define the right hop pointer in each region. Note that some of relationships between ranges are an enclosure. That means that any range which includes another range can not cover up their entire region because of the longest prefix match. For example, the range from 101010 to 101011 is not the range of 101000 any more.

However, a prefix is a unique string not a region. To cover a range using a single prefix, two hop pointers are required to be associated with the prefix. Figure 2 shows the final binary table including the = pointer and the > pointer. The = pointer is returned if incoming addresses are exactly equal to the corresponding entry in the table. The > pointer is the next hop that has a relation with addresses that are greater than the corresponding entry. Two pointers are sufficient to cover up their interval. If the address 101001 finds its next hop in the table, binary search will end up at the second entry and return P2 as a next hop because 101001 is greater than the second entry and less than the third entry in the table.

		=	>
P1)	1 0 0 0 0 0	P1	P1
P2)	1 0 1 0 0 0	P2	P2
P3)	1 0 1 0 1 0	P3	P3
	1 0 1 0 1 1	P3	P4
P4)	1 0 1 1 0 0	P4	P4
	1 0 1 1 0 1	P4	P2
	1 0 1 1 1 1	P2	P1
	1 1 1 1 1 1	P1	-

Figure 2. The binary search table after pre-computation.

Prefix range search has an advantage that applies binary search to IP address lookup even though a prefix is a variable length. However, this approach requires pre-computation for building a binary table and calculating two pointers [2]. And the required memory size is larger than for other algorithms. There is a two-fold increase of in the memory size in the worst case for adding the high entries. Sufficient memory for the = and > pointers is also required.

3 The proposed algorithm

Prefix range search requires twice the number of original prefixes in the worst case. This approach also uses two pointers every prefixes as well. In this section, we describe the proposed algorithm to solve these problems.

3.1 Motivations

The advantage of prefix range search is found in using binary search. To do binary search we also apply a concept of a range to prefixes. Like the prefix range search, prefixes are expanded and padded with 0 and 1. These padded strings are then sorted. It follows the same process of the proposed method until all the prefixes are sorted.

However, in the prefix range search, both the high entry and the low entry should be connected to each other.

In other words, one range means that the intervals have a starting point and an endpoint. By contrast, if an entry for starting point is only used for a range, this entry has an infinite range from itself. But if the range meets another starting point, the range from the previous starting point is finished before the new starting point. This is an important fact for our algorithm. This approach can reduce two pointers of prefix range search to one pointer. The differences are depicted as two vertical axes in Figure 3. The figure on the left expresses the conventional idea that cuts off the line (or range) using two points. The problem is that all entries must simultaneously have both a starting point and an endpoint, except for the top and bottom entries in the list. As a result, the endpoint for the previous entry can be the starting point of the next entry. Consequently, all prefixes require the = and > pointers.

In the right figure, all entries have only starting points as opposed to the left. It makes the = and > pointers to be put together into the ≥ pointers. Accordingly, the memory size for pointers can be reduced by half.

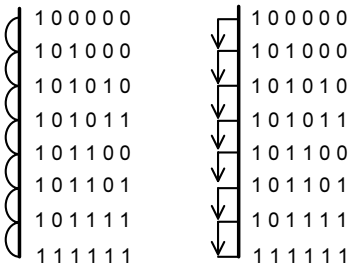

Figure 3. Two types of representation for a range.

3.2 Encoding prefixes by the proposed algorithm

There are some limitations when the expression on the right in Figure 3 is applied to the all entries. At the low entries, the value of the two pointers is the same. On the contrary, the high entries have two different values in pointers. In Figure 2, the prefix 101101 gives P4 and P2 for the = pointer and the > pointer, respectively. These pointers can not be mixed up into the ≥ pointer without modifications.

The solution for this problem is to add the value 1 to the high entry, because then all prefixes would have the same value on two pointers so that only one pointer is possible to be used. Because the high entry added by 1 does not need a starting pointer any more. In the previous method, since the high entry is made from the same prefix that makes the low entry, the high entry has to remember the corresponding pointer, the low entry.

There is another effect from adding the value 1 to the high entries. It is to reduce the number of entries in the binary search table. In Figure 2, the address 101011 and 1 makes the address 101100 which is the next address to the address 101011. In general, if the low entry is next to the high entry, the high entry plus 1 equals the low entry. Therefore, the high entry can be eliminated. Figure 4 shows the final binary table with some modified prefixes. The number of entries is reduced by 1 and the number of pointers is reduced by half. In addition, all high entries are added by 1.

		≥
P1)	1 0 0 0 0 0	P1
P2)	1 0 1 0 0 0	P2
P3)	1 0 1 0 1 0	P3
P4)	1 0 1 1 0 0	P4
	1 0 1 1 1 0	P2
	1 1 0 0 0 0	P1
	1 0 0 0 0 0 0	-

Figure 4. The proposed binary search table.

In the proposed binary search on range, the method where the high entries are added by 1 reduces two types of memories. The first thing is that it put the = and > pointers together into the ≥ pointers. Second is that if the low entry is the same with the high entry added by 1, the high entry is removed.

3.3 Considering the prefix with 32 bit length

Mixing up two pointers into one pointer is likely perfect. But there is a problem the prefix which is 32 bits in length can not be a range. This is because the prefix with length 32 is an only one point in an environment using 32 bit length on IP addresses. In order to adapt to an entry with a single pointer, one entry to which prefix with length 32 plus one is added in the table. The added entry has a role of a starting point like the other entries. It makes prefixes with length 32 expand their range.

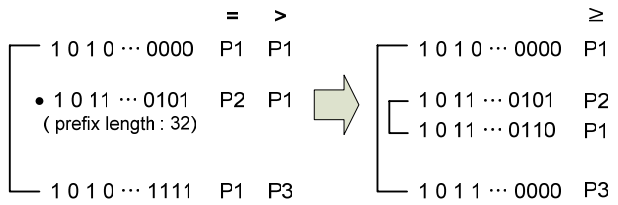

Figure 5. The modification for a prefix with 32 bit length.

Figure 5 is the result of modifying prefixes with length 32. In this example, a entry with prefix P2 is length 32 included in P1. If the entry uses only the prefix value P2 for the ≥ pointer, the larger address than P2 will be P2 as their next hop although P1 is right hop. To prevent the incorrect results from the prefix with length 32, new entry(1011···0110) is added. Hence, the entry with length 32(1011···0101) have been changed from the point to the

range. Finally, we are able to maintain the reduced pointers in our algorithm

4 Simulation results

The prefix range search and proposed algorithm were simulated with actual data having different prefixes. We compared the number of entries and the number of pointers for each data. Figure 6 shows the comparison of the number of entries. The proposed approach reduces the number of entries by about 20 % on the average. As the number of prefixes grows, the difference of the entry requirement between prefix range search and proposed algorithm is increased and the difference between the number of the prefixes and the number of the proposed entries is decreased.

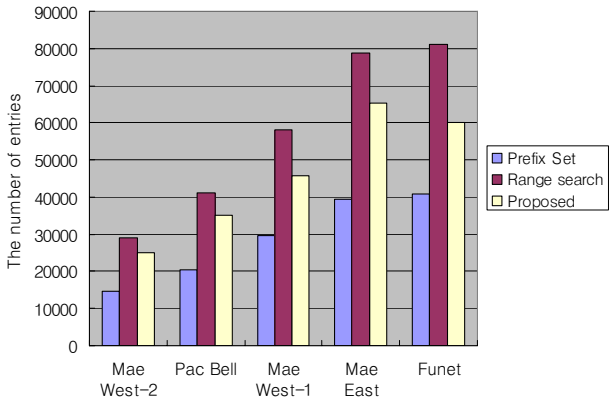

Figure 6. The comparison of the number of entries.

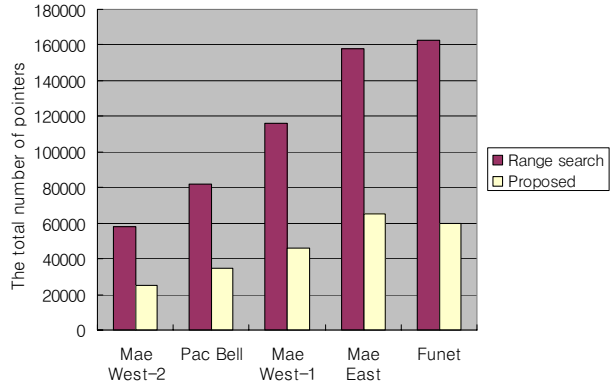

Figure 7. The number of pointers in two algorithms

The comparison of the number of pointers is shown in Figure 7. The larger the number of entries, the greater the difference of the total number of pointers is. Because prefix ranges search uses two pointers on every entry.

In the two comparisons above, the proposed algorithm gives the better performance. We expected that the proposed algorithm would have had a little increase in the number of entries when empty space between ranges does not exist. On the contrary, in a prefix range search required the number of entries will be increased by a rate of increase in the prefixes. Besides, the number of pointers grows twice the number of entries.

5 Conclusions

Binary search is a good method for an exact match. Unfortunately, it can not be applied to IP address lookup directly since the prefixes are a variable length and a hierarchical structure. A prefix range search has been solved that the IP address lookup is performed with binary search. However, the prefix range search requires twice the number of entries and two pointers on every entry. Hence, it needs more memory than other algorithms for IP address lookup. This paper solved this problem. The proposed algorithm reduces the number of entries and pointers. The simulation results show there is a decrease of 20 % in the number of entries than for previous algorithms.

References

[1] George Varghese, *Network Algorithmics*, Morgan Kaufmann, 2005.

[2] C. Yim, B. Lee, and H. Lim, "Efficient Binary Search for IP Address Lookup", *IEEE Communication Letters*, Vol. 9, No. 7, pp. 652-654, July 2005.

[3] B. Lampson, V. Srinivasan, and G. Varghese, "IP Lookups Using Multiway and Multicolumn Search", *IEEE/ACM Trans. Networking*, Vol. 7, pp. 324-334, June 1999.

[4] H. J. Chao, "Next generation routers," *Proceedings of the IEEE*, vol. 90, pp. 1518-1558, Sept. 2002.

[5] M. A. Ruiz-Sanchex, E. W. Biersack, and W. Dabbous, "Survey and taxonomy of IP address lookup algorithms", *IEEE Network*, Vol. 15, pp. 8-23, March 2001.

[6] N. Yazdani, and P. S. Min, "Fast and scalable schemes for the IP address lookup problem", Proc. IEEE HPSR, pp. 83-92, 2000.

[7] L. C. Wuu, K. M. Chen, and T. J. Liu, "A longest prefix first search tree for IP lookup", Proc. IEEE International Conference on communications, pp. 989-993, May 2005.

[8] S. Suri, G. Varghese, and P. R. Warkhede, "Multiway Range Trees: Scalable IP Lookup with Fast Updates", Proc. IEEE GLOBECOM, pp. 1610-1613, November 2001.