A New Efficient IP Address Lookup Based on Binary Search

Hyun-Sik Kim, Hyuntae Park, Daein Kang, Sungho Kang
Yonsei University, 134 Shinchon-dong, Seodaemoon-gu, Seoul, 120-749, Korea
Tel.: (02)2123-2775, E-mail: hskim@soc.yonsei.ac.kr

Abstract

IP address lookup is an important function in a router to settle overall performance of the router. Binary search on range is a useful algorithm based on binary search for IP address lookup. However, this approach requires a lot of memories for adapting binary search to variable-length prefixes. We propose the improved algorithm to reduce memory requirements than binary search on range dramatically by a different view of a range. It reduces the number of entries and the memory requirements for pointers by about 21% and 52% on average, respectively.

Key Words — IP address lookup, longest prefix matching, binary search, binary search on range

I. INTRODUCTION

Recently, diverse equipments are being connected on network environment such as a cellular phone, PDA, and television as well as computer systems. This situation is strongly advised to have an efficient communication between end systems on the Internet networks.

A router is an important part to connect the networks and affect overall network performance. The major role of the router is to process a packet header in general network processors [1] that operate IP address lookup, packet classification and additional packet modification. The router has two kinds of problem such as bandwidth and the size of a lookup table [2].

Due to developments of a wire technology, backbone links have been upgrading to 40Gbps. The router must process an incoming packet within 8ns (8*40byte / 40Gbps) for a 40-byte packet, in order to meet such a wire speed [3]. As the size of a lookup table increases, IP address lookup is getting slower, since IP address lookup is related to memory access. To perform IP address lookup, it is necessary to access external memory such as SRAM, DRAM or SDRAM. Network processors make complex operations on data from the memory. Thus, IP address lookup has become an important issue in routers and essential consideration for overall performance of the routers.

There are two methods for lookup such as an exact match and longest prefix match. The exact match is to find an output port associated with a prefix in a lookup table which is exactly the same as a packet destination address. The exact match gives guaranteed lookup that can be done at high speeds [4]. Whereas, longest prefix match is to find the output port as a longest matched result in the lookup table from the destination address. However, the Internet uses variable-length prefixes so that longest prefix match should be employed although the exact match is faster than longest prefix match for IP address lookup.

A lot of algorithms based on longest prefix match for IP address lookup have been studied [2][5][6]. On the other hand, some algorithms using binary search are also proposed. Binary search is a well-known technique and useful for the exact match. Besides, it is free of a restriction on patents so that vendors can apply it into diverse applications without any permission [2]. But, binary search cannot be used for IP address lookup directly, because of variable-length prefixes. In some papers, binary search have been used as assistance for IP address lookup. Among them, unlike other algorithms, binary search on range [4] applies it as a main method for IP address lookup by extracting a concept of a range from prefixes in order to achieve good search time. However, this algorithm requires memory up to twice the number of prefixes in the worst case. This paper deals with such a problem and proposes new algorithm based on binary search.

II. PREVIOUS RESEARCH

Since a destination address of an incoming packet does not carry with any information on a prefix length, IP address lookup requires an efficient algorithm to look up all entries in a lookup table by longest prefix match. A basic algorithm is a trie method [7][8][9], which is a simplest technique to find a next hop with each node containing a 0-pointer and 1-pointer to its child nodes. The trie is simple and natural to represent a prefix. Special hardware [10][11] can be used for IP address lookup by focusing on memory architecture prefix information stored in.
Most schemes focus on longest prefix match, on the other hand, binary search on range is based on an exact match. Binary search on range starts from a good approach that a prefix can actually be thought of as a range. The prefix has its own space on a dimension of IP address. This makes it possible to adapt variable-length prefixes to the exact match.

III. THE PROPOSED ALGORITHM

As binary search is an efficient technique to reduce the number of memory accesses, binary search on range uses binary search for looking up entries in a lookup table. Although binary search on range gives a new approach that a prefix can change into a range for employing binary search, it requires twice the number of prefixes in the worst case. This paper proposes the new improved algorithm based on the range to deal with the problem.

To show how to make a lookup table by extending prefixes to a range, we used a simple set of 4 prefixes in Table 1. A prefix 001* represents a range from 001000 to 001111. Hence, one prefix extends two entries, a start-entry padded with 0 and an end-entry padded with 1. In this paper, the prefix padded with 0 and 1 are defined a low entry and high entry, respectively.

<table>
<thead>
<tr>
<th>Prefix</th>
<th>Next Hop</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>0*</td>
</tr>
<tr>
<td>P2</td>
<td>001*</td>
</tr>
<tr>
<td>P3</td>
<td>00101*</td>
</tr>
<tr>
<td>P4</td>
<td>01011*</td>
</tr>
</tbody>
</table>

To show how to make a lookup table by extending prefixes to a range, we used a simple set of 4 prefixes in Table 1. A prefix 001* represents a range from 001000 to 001111. Hence, one prefix extends two entries, a start-entry padded with 0 and an end-entry padded with 1. In this paper, the prefix padded with 0 and 1 are defined a low entry and high entry, respectively.

In order to represent the range using a single prefix, two hop pointers are required to be associated with the prefix. Fig. 1 shows the final lookup table including a = pointer and a > pointer. A final entry is to be found via binary search, and then the = pointer will be returned if a destination address is exactly equal to the corresponding entry in the lookup table, or the > pointer is the next hop that has a relation with the destination address which are greater than the corresponding entry. Two pointers are sufficient to cover up its interval. For example, if an address 001001 finds its next hop in the lookup table, binary search will end up at the second entry and returns P2 as a next hop. It has an advantage to apply binary search to IP address. However, this approach requires up to double the number of original prefixes as well as sufficient memory for the = and > pointers.

Fig. 2 shows a difference between binary search on range and the proposed algorithm with horizontal axes representing a range. Fig. 2(a) shows a horizontal axis each prefix is occupying their region, rotated of the lookup table in Fig. 1. It is a change from prefixes to a range in which a lined arrow is a low entry and a dotted arrow is a high entry. Fig. 2(b) shows the range between two entries without a connection to the end point for an intensive approach. All end entries are changed to the starting point of a range like a low entry, so that high entries become the starting point having their region, not an end of the connection with the low entry. Accordingly, it looks like only one pointer ≥ is sufficient to represent its range.

B. Build a lookup table

There are some limitations when a new expression is applied to all entries. At a low entry, the value of two pointers is the same. On the contrary, a high entry has two different values in pointers, since a > pointer represents a part which is greater than its prefix. For example, an entry 001111 has two P3 and P1 pointers. For that reason, these pointers can not be mixed up into a ≥ pointer without modifications.
To apply what Fig. 2(b) illustrates completely, a modification to add value 1 to a high entry is going to be used, because the high entry itself is included in a region of a low entry. Therefore, a = pointer of the high entry does not need to be exist. Instead, the high entry added by l maintains a region of a > pointer. Finally, it is possible to put the = and > pointers together into the ≥ pointer.

We could find another impressive effect during applying this method. Note that no an entry 000111 is needed any more, because adding value 1 to the entry 000111 becomes 001000 which is the same as the next low entry 1 value off. As a result, all high entries connected with the next low entry continuously can be eliminated. The other high entries are required to support discrete regions. The final result from the modification is shown in Fig. 2(c). We can easily get a lookup table by rotating this final result. After all, adding value 1 to all high entries from a different view of a range leads two results that two pointers are mixed up into one pointer and the number of entries reduces

C. Insert a flag bit for 32 bit-length prefixes

It is important to process a 32-length prefix effectively for a trend of growing the 32-length prefix. The proposed algorithm cannot be directly applied to the 32-length prefix.
IV. SIMULATION RESULTS

We have simulated binary search on range and the proposed algorithm with 7 prefix sets. We have compared the number of entries and memory requirements. Fig. 5 shows the comparison of the number of memories. Binary search on range and the proposed algorithm has an increase of 82.84% and 45.42% on average, respectively, from the original prefix sets. For two data of Port80 and Telstra, the proposed algorithm is analogous to the number of prefixes. It means that the difference of the number of entries from binary search on range will be more increased as larger the size of a prefix set. Thus, the proposed algorithm reduces memory requirements.

Memory size for pointers has been calculated, including a flag bit in the proposed algorithm. The proposed algorithm decreases the memory size by 52.32% on average. Fig. 6 shows the comparison of the memory size for the pointers and the flag bit. The reason for that a reduction rate is over 50% though only one pointer is eliminated is a decrease of the number of entries.

V. CONCLUSIONS

In general, an exact match is more efficient than longest prefix match. While algorithms based on longest prefix match are presented, binary search on range apply binary search to a variable-length prefix. Binary search can be easily implemented in general network processors and bounds the worst-case search time of $O(\log_2 N)$, where $N$ is the number of entry. However, binary search on range is in a place, where it requires twice the number of entries and two pointers on every entry. This paper proposes the improved algorithm to solve this problem with a combination of two $=$ and $>$ pointers into a $\geq$ pointer. The comparisons show the proposed algorithm gives a better performance. It reduces the number of entries as well as the number of pointers.

REFERENCES