A Partitioned Binary Search Scheme on Multiple Trees for Efficient IP Address Lookup

Hyuntae Park             Sungho Kang
Dept. of Electrical and Electronic Engineering
Yonsei University
Seoul, Korea
{radiob, shkang}@yonsei.ac.kr

Byung In Moon
School of Electrical Engineering
& Computer Science
Kyungpook National University
Daegu, Korea.
bihmoon@knu.ac.kr

Abstract – IP address lookup is becoming critical because of increasing entries in a forwarding table, line-speed and traffic in the Internet. Binary search is the most popular scheme based on S/W implementation. This search scheme has problems according to the parent-child relationship of prefixes. In this paper, we proposed the partitioned binary search scheme to split the parent-child relationship. All prefixes were classified as disjoint prefixes, enclosure prefixes, and child prefixes, and then were stored and searched separately. Performance results show that the required memory size is 151Kbytes to store about 30K entries and an address lookup on average is achieved by 8.23 memory accesses.

Keywords: IP address lookup, partitioned binary search, disjoint prefix, enclosure prefix, child prefix.

1 Introduction

High-speed Internet service techniques such as DSL, cable modem, and gigabit Ethernet are improving and the number of hosts on the Internet is growing exponentially. New multimedia applications require higher bandwidth as well. Accordingly, the growth of traffic on the Internet is bowing up. Fortunately, optical transmission technologies provide high bandwidth. It can transmit data in several gigabits; 10Gbps in OC192 and 40Gbps in OC768c have become possible recently. In order to keep pace with this speed and bandwidth, the packet forwarding that achieves the same speed is required. As the forwarding table in routers is growing rapidly larger, it is difficult to perform IP address lookup processing in a fast line-speed for all incoming packets. In order to reduce forwarding table entries, Classless Inter-domain Routing (CIDR) is deployed to allow for arbitrary aggregation of network addresses. Because the prefixes have arbitrary lengths in CIDR, the most specific matching among matching entries with various lengths, which is the longest matching prefix (LPM) is required. Therefore, when using CIDR, the search in a forwarding table can no longer be performed by exact matching for class-based addressing. Because of the complexity of this operation, it becomes a major bottleneck of router performance and one of the more critical issues for high-speed networking techniques [1].

2 Previous schemes

Binary trie is the most popular data structure and a search scheme for IP address lookup. In a trie, prefixes are stored in a node defined by the path from the root. Searching based on trie can be performed by a comparison of each bit of the destination address through a trie path in order. This scheme has empty internal nodes such that the required memory size is larger than for other schemes. As well, the number of memory accesses in worst case is same as a length of IP address [3]. It is an unfeasible performance. The binary search on range [4] converges each prefix to an interval which has a start point with prefix plus padding 0 to an end of address and an end point with prefix plus padding 1. Accordingly, all prefixes have the same length which is equal to the maximum length of address. A match pointer and a larger pointer are precomputed for each point. A match pointer becomes the best match prefix (BMP) during a binary search. Using these pointers, perfect binary search is performed on intervals of prefixes. However, a pre-computation for pointers is required and the number of intervals in the worst case is twice that of the number of prefixes.
Binary prefix tree (BPT) has been proposed [5]. It can be binary searching by sorting and comparison on prefixes with variable length. A forwarding table using tree is composed without internal empty nodes such that the memory size is more efficient than that of trie based lookup schemes. Unfortunately, this tree is very unbalanced. In order to solve this problem, a weighted binary prefix tree (WBPT) is proposed [6]. In this scheme, which considers the number of descendents to select an entry in each node, a more balanced and shorter depth tree than BPT is achieved.

On the other hand, because BPT and WBPT have greater depth because of the parent-child relationship of prefixes, multiple balanced prefix tree (MBPT) is proposed [7]. If prefixes are fully disjointed, a perfect balanced tree can be constructed. By apply this principle; the parent-child relationship of prefixes is separated such that a forwarding table is composed of multiple balanced trees. However, multiple sub-trees are overlapped by consecutive parent-child relationship. Therefore, the performance of MBPT depends on the aggregate level of incoming packets.

3 The Proposed Scheme

An IP address lookup scheme using a binary search is the most popular scheme among software based packet forwarding systems because it has a fast lookup speed, \( O(\log N) \). However, there are three problems in applying binary search for LPM because searching domains for LPM are both prefix values and lengths. First, sorting is required according to variable length values. Second, a misleading match is due to the parent-child relationship of prefixes. If a child is found earlier than corresponding parent, the parent prefix cannot be found. In other words, a parent prefix is placed on the upper node rather than a located node of child prefix. Therefore, a comparison of prefix values as well as the parent-child relation should be considered in the sorting procedure. Third, even though the specific prefix is matched in the middle of tree, searching should go on through a corresponding path to a leaf node. Although the shorter prefix is matched, another longer prefix can be present, which is also due to the parent-child relationship.

In our proposed scheme, in order to solve problems mentioned above, we proposed the partitioned binary prefix tree using multiple sub trees. The goal of the proposed partitioned tree is to split the parent-child relationship. Accordingly, a disjoint prefix set without dependency, a parent prefix set called an enclosure with minimized relationship, and a child prefix set with the disjoint feature are handled separately. Thanks to disjoint and child prefix sets that no longer have another prefix, it constructs a perfectly balanced tree such that it can be a perfect binary search. If any prefix is found, the searching operation can be dropped at once because another, longer prefix cannot exist. Therefore, it has the advantages that it is an efficient memory size because left and right pointers are not required for a trace of a tree and is a fast searching time, \( O(\log N) \).

On the contrary, an enclosure prefix set is handled with more complexity. Because an enclosure set has the parent-child relationship, it should be constructed like an unbalanced tree. In this tree, the parent prefix should be properly located on a higher node than its child prefixes. In spite of that, an enclosure prefix set is a pure prefix set with the parent-child relationship expected disjoint prefixes such that the number of these is reduced to a minimum. Therefore, the difficulty in handling enclosure prefixes is minimized in the proposed scheme.

3.1 Prefix Sorting

The terminologies of this work for comparison and sorting of prefix refer to definitions in [5]. However, the parent-child relationship is newly defined as follows.

Definition 1 Disjoint
Two prefixes are not matched.
Ex) For \( A=1011 \) and \( B=111 \), \( A \) and \( B \) are disjoints.

Definition 2 Enclosure
If there exists at least one element in the data set such that \( A \) is a prefix of that data element, a prefix \( A \) is called an enclosure.
Ex) For \( A=1011 \) and \( B=101100 \), \( A \) is an enclosure of \( B \).

Definition 3 Child
If a prefix \( A \) is an enclosure of \( B \) and a prefix \( B \) is not an enclosure of other prefixes, a prefix \( B \) is called a child of \( A \).
Ex) For \( A=1011 \), \( B=101100 \), \( C=101101 \), and \( D=1011010 \),
\( A \) is an enclosure of \( B \) and \( B \) is a child of \( A \).
Although \( A \) and \( C \) have child relation, \( C \) is not a child of \( A \) because there is a child of \( C \).
Merely \( C \) is an enclosure of \( D \) and \( D \) is a child of \( C \).

All prefix sets in a forwarding table are primarily sorted by definitions of [5]. Then, they can be classified as disjoint prefixes, enclosure prefixes, and child prefixes by our definitions. Each prefix has a unique type. Enclosure prefixes are leveled by a position within a child relationship. Child prefixes on same level with the same enclosure have a feature of disjoint. Namely, Sub trees composed of child prefixes with a same enclosure can be perfect balanced tree. For the given set of prefixes, sorted and classified prefixes are shown as Fig. 1.

3.2 Building

Sorted and classified prefixes are stored into triple forwarding tables according to their particular types such as disjoint, enclosure and child. Sorted disjoint prefixes are stored into the disjoint table and sorted child prefixes are stored into the enclosure sub table. Because these are
built as perfectly balanced trees, these only have prefix and output port information. In the enclosure sub trees, prefixes are stored except the overlapped part of prefixes. Enclosure prefixes are stored into an enclosure main table. 1st level enclosures are the first candidate for building. If an allocated enclosure prefix has its child prefix which is the next level enclosure, its child prefix becomes newly candidate. This process is repeated until there are no more prefixes to be stored. Each node is connected by left and right pointers according to a size of values. As well, if an enclosure prefix at each node has a corresponding child prefix set, it is connected to an enclosure sub table by sub pointers. We based on our proposed scheme on an example prefix set used in Fig. 1, for a example, Fig. 2 shows the proposed partitioned binary trees with a 1-bit range table.

Entry structures of each table are shown as Fig. 3. In order to reduce the number of memory accesses in a binary search in the disjoint table and the enclosure main table, we apply the additional searching method with a range table. It is a popular method in a binary search for LPM based on no existence of prefixes less than 8-bit length. In the range table, each entry has a pointer to the starting index of prefixes and the number of prefixes included corresponding ranges. In order to shorten the first 8-bit length of prefix, this table has $2^8$ entries.

### 3.3 Searching

Incoming packets are searched through the disjoint table and the enclosure main table in parallel. Firstly, the search space of both tables is decided by the index of the range tables using the first 8-bit of destination addresses. In the disjoint tree, perfect binary search is performed. If the matched prefix is found and no match is found, search is completed. Simultaneously with this, in the enclosure main tree, the search is continued to the leaf node through corresponding path. In the middle of the search trace, whenever a matched prefix is found, the best match prefix (BMP) is updated. When the search trace is done, the sub tree of BMP is searched. When the matched prefix is found in corresponding enclosure sub tree or there is not the sub tree, the search is completed. If there is no match in this searching, the output pointer of BMP is returned. In other cases, the output pointer of the matched prefix is returned. Because the result of search exists in either the disjoint tree or the enclosure tree, there is the only returned output pointer. For example of address 111011, 1’s range is decided firstly. It is not matched in 1’s disjoint tree. In 1’s enclosure tree, this example prefix is larger than 10 and 1011. Then, 111 is matched. It is memorized by BMP and the search is going on. The next matched prefix is 11101 and the BMP is updated by this. When the search is completed in the enclosure main tree, BMP is 11101. Hence, the sub tree of 11101 is searched, but it is not matched. Finally, the output pointer of 11101 is returned and the search is wholly completed.

![Figure 1. Sorted and classified prefixes.](image1)

![Figure 2. The proposed partitioned binary trees.](image2)

![Figure 3. The entry structures in each table.](image3)

### 3.4 Incremental Updating and Scalability

The proposed scheme is available for an incremental updating. Updating is the same operation with the searching. According to this, the type of inserted prefix is decided. It is allocated into the proper table. If this type of previous prefix is changed by the relation with the inserted prefix, both the affected previous prefix and the inserted prefix are reallocated according to the changed type. In this specific case, a disjoint can be transmitted to an enclosure, and then the partial part of enclosure tree can be re-constructed. Besides, the extension by increasing entries and applying IPv6 is available. Depending on increased incoming entries, the number of entries in tables is simply added without more complexity. As a result, the lookup speed performance is not worse. As well, in order to apply IPv6 the extension of the prefix
field is merely required. Therefore, the proposed scheme has a good scalability.

4 Performance Comparison

The performance of the proposed scheme is evaluated using five routing data from real backbone routers. Fig. 4 and 5 show the performance comparison with binary trie[3], range search[4], BPT[5], and WBPT[6] in terms of the average number and the maximum number of memory access. Partitioned binary tree in the proposed scheme improve the performance of a lookup speed shown as Fig. 4 and 5. The reason for improvements is that previous binary search schemes have single plump tree, whereas the proposed scheme have proper multiple trees. To indicate a weak point of the proposed scheme, the maximum number of memory access may be worse if there is a lot of enclosure prefix set like FUNET. Nevertheless, the proposed scheme has a good performance over all. Moreover, we almost observe that the portion of enclosure prefix set is about 10%. Besides, the required memory size is reduced as Fig. 6. Because a disjoint set and an enclosure set are separated in the proposed scheme, unnecessary pointers can be reduced efficiently. Although triple tables are required, the performance of the proposed scheme is faster lookup speed and less memory space than previous schemes.

Figure 4. Comparison of the average number of memory accesses.

Figure 5. Comparison of the maximum number of memory accesses.

5 Conclusions

In this paper, we proposed the partitioned binary search scheme to split the parent-child relationship. All prefixes are classified as disjoint prefix, enclosure prefix, and child prefix. Then, these are sorted for binary search and stored and searched separately. Performance results show the improvement of our proposed scheme than previous binary search schemes for LPM. The required memory size is 151Kbytes to store about 30K entries and an address lookup in average is achieved by 8.23 memory accesses.

Acknowledgments

This research was supported by University IT Research Center Project of Korea.

References


