An IPv6 Address Lookup Algorithm Based on Multi-bit Trie and Prefix Hierarchy

Mei-gen Huang and Yong-zai Si

ABSTRACT

The size of IPv6 routing table growing rapidly, fast ip address lookup is becoming critical in high-speed packet forwarding. However, most of the IPv4 routing algorithm is not suitable for IPv6, measure the characteristics of IPv6 and the routing table address prefix hierarchical distribution. In this paper, a new fast IP address lookup algorithm based on multi-bit trie and prefix hierarchy is proposed. The proposed algorithm has good scalability and effectively reduces the depth of the search tree.

INTRODUCTION

With the rapid development of new technologies such as Internet of Things, big data and cloud computing, the demand for IP addresses is increasing. However, the number of IPv4 addresses is almost exhausted. Meanwhile, the number of Internet users are increasing and the number of IPv4 addresses is far from to meet the needs of the market. IPv6[1], the next-generation address protocol, has a 128-bit and is large enough to meet the need for new technologies.

IPv6 solved the quantity of Internet users, but the address length of IPv6 has also brought new challenges to the routing algorithm. Both IPv6 and IPv4 are using longest matching prefix (LMP), so IPv6 can inherit part of the IPv4 routing algorithm. But, in the algorithm performance, the differences between IPv6 and IPv4 are as follows. Firstly, IPv6 address length is four times than IPv4, if IPv6 directly use IPv4 routing algorithm will result in increased search time and space complexity. Secondly, IPv4 routing scale is stable and IPv6 routing table in rapid growth stage. According to the statistics of AS 2.0 BGP[2] routing table, the number of IPv6 routing tables increases exponentially from 2010 to 2016, and the number of routing tables increases from 2,500 to 33,861. With the rapid growth of the routing table, the routing table updates more frequently. Thirdly, IPv6 prefix length has a certain concentration and mainly distributed in 32, 40, 48 and 64, there are a small part of the prefix is greater than 64. At length, IPv6 prefix distribution is...
also centralized. As shown in [3], IPv6 global unicast address allocation uses an address range beginning with 001 (200 :: / 3).

In this paper, a new IP address lookup algorithm using multi-bit trie and prefix hierarchy is proposed. The data structure of the algorithm is divided into two parts. The first part is stored with 4-bit trie, and the second part is based on prefix hierarchy. The organization of this paper is as follows. In Section 2, we briefly summarize the related works for IP address lookup. Section 3 presents our proposed algorithm. Finally concludes the paper.

RELATED WORK

BSR[4] 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. The BSR algorithm uses two nodes to represent prefixes, so a maximum of two times the number of prefixes is required. The prefix in routing table have containing and contained relationship. A routing prefix may be a subset or parent of another routing prefix. Figure 1 illustrates the relationship of the prefix, P7 is the parent node of P8, then prefix P7 containing P8, and P8 is contained by P7. The depth of the prefix is the number of real nodes passing from the root node to the destination prefix. For example, the level of P1 is 0, the level of P3 is 1, and so on.

![Figure 1. The relationship of the prefix.](image)

Binary Search Algorithms Based on Trie

A binary trie is the most intuitive and easy data structure for the IP address lookup. Each node contains a maximum of two nodes and represents the prefix of the next node. In the binary trie, some nodes contain prefix information. The time complexity of insert and update is $O(W)$. But, the binary trie is not balanced and the depth of the trie is usually $W$, where $W$ is the maximum prefix length. Moreover, there are many blank node in the binary trie, resulting in waste of memory space. In order to solve the problem of blank nodes and educe the number of comparisons, K. Sklower et al. [5] used PATRICIA to compress blank nodes in the binary trie. But it is mainly for small-scale routing table, the time and space complexity are $O(W)$ and $O(NW)$. Sahni S et al.[6] used multi-bit trie to integrate k-bit into a node. The time complexity of this algorithm is $O(W/k)$, which effectively improves the search speed and reduces the depth of the trie. Lim H et al.[7] used priority trie to reverse the binary trie structure. This algorithm give priority to the longer prefix and remove the empty space in the trie.
Algorithms based on hashing

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. In addition, Song H et al.[8] proposed to use Bloom Filter to longest prefix match in routing table. This algorithm provides efficient and fast lookup with less memory, but with false positives situation. However, hash-based schemes have the limitation of hash collision and it is not easy to find a suitable hash function.

In other work, Waldvogel M et al.[9] proposed a binary search method based on prefix length. This method has $O(\log W)$ search complexity, but requires a large number of initialization and complex update process. It will also produce a large number of Markers and increase the computational complexity and memory space. In [10], the number of Markers is reduced by a complex algorithm, and the memory consumption of the whole algorithm is reduced. This method is based on hashing, which will inevitably produce hash conflict and affect the algorithm efficiency and practical effect.

PROPOSED ALGORITHMS

From the Potaroo IPv6 backbone network BGP routing table, we can see that the top 16-bit of IPv6 are 2001, 2400, 26 **, 280 *, 2a0 *, 2c0 *, which show that the top 16-bit of IPv6 are basically fixed. The top 16-bit of IPv6 in Potaroo shown in Table 1. As can be seen from the analysis of section 2, the top 16-bit IPv6 is relatively fixed, in which the first four bits (0010) remain unchanged. The first four to seven also only six kinds of values, the difference lies in the latter 8-bit address. So these data need to be processed specially.

<table>
<thead>
<tr>
<th>Types</th>
<th>Potaroo</th>
<th>Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>2001</td>
<td>943</td>
<td>4.93</td>
</tr>
<tr>
<td>240*</td>
<td>3163</td>
<td>16.52</td>
</tr>
<tr>
<td>26**</td>
<td>7026</td>
<td>36.69</td>
</tr>
<tr>
<td>280*</td>
<td>2572</td>
<td>13.43</td>
</tr>
<tr>
<td>2a0*</td>
<td>5157</td>
<td>26.93</td>
</tr>
<tr>
<td>2c0*</td>
<td>286</td>
<td>1.49</td>
</tr>
</tbody>
</table>

Table 1. Top 16-bit of IPv6 in Potaroo.

Basic Mechanism of Our Scheme

Multi-bit trie structure not only has good time complexity, but also ensures the expansibility of algorithm. So the first part we use a multi-bit trie construct.

Set the starting point of the node with the same top 16-bit prefix address as the second part data to rooti. The prefixes starting from rooti are divided into disjoint sets and set to S0th. Each of the prefixes in the S0th maintains a set and points to all the prefixes contained by the prefix, set to S1th and divides all its prefixes into disjoint sets. Likewise, each prefix in the S1th maintains a set and points to all prefixes containing by the prefix, set to S2th and divides all its prefixes into disjoint sets, and so on. All prefixes are processed until there is no prefix can be divided.

The data structure of prefixes shown in Table 2 is constructed as follows. All the prefixes in the figure are IPv6 with 2001 as the top 16-bit. The S0th contains P1 and P2, the set A contains P1, and the set B contains P2. The S1th contains P3, P5, P6, P7, P12. The set C contains P3, P5, the pointer from set A to the C, the set D
contains P6, P7, P12, the pointer from set B to set D. S2 contains P4, P8, P9, P10, P11, set E contains P4, the pointer from set C to set E, set F contains P8, P9, the pointer from set D to set F, set G contains P10, P11, the pointer from set D points to set G.

### Table 2. Data Structure of prefixes set.

<table>
<thead>
<tr>
<th>Prefix</th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
<th>P4</th>
<th>P5</th>
<th>P6</th>
<th>P7</th>
<th>P8</th>
<th>P9</th>
<th>P10</th>
<th>P11</th>
<th>P12</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>0*</td>
<td>1*</td>
<td>01*</td>
<td>011*</td>
<td>001*</td>
<td>111*</td>
<td>P7</td>
<td>1101*</td>
<td>1101*</td>
<td>1011*</td>
<td>1101*</td>
<td>10111*</td>
</tr>
<tr>
<td>P2</td>
<td></td>
<td>11101*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Algorithm Data Structure Construction

The first part stores the top 16-bit of the prefix, stores it with 4-bit trie and the leaf node of the tree acts as the root node of the second part structure.

The second part we choose B-tree which is a balanced tree, can effectively control the depth of the search tree, and improve B-tree structure. A FLAG field is defined for each prefix address to identify whether the key is greater than or less than the prefix.

```c
struct BTreeSlevell {
    int num; //The number of elements in the set
    T *K;   //array points to a set
    long value; //remove the high 16-bit prefix
    BTreeSlevell <T> *parent;  //point to parent
    BTreeSlevell <T> **child;  //point to the children node array
    bool FLAG;  // 1 is greater than the prefix, and 0 is less than the prefix
}
```

### Lookup in Our Proposed Algorithm

As Fig.2 illustrate, the top16-bit of the prefix are searched in the 4-bit trie. If there is no match, search fail. If exist, the search is made in the B-tree constructed with the top 16-bit of the prefix. When the search prefix (after removing the top 16-bit) belongs to P9. First, query in set A, P2 matching success and point to set B, search in set B, P6 successfully matched and point to set D, search in set D, P9 is returned as the longest match prefix.

![Figure 4. Lookup in Data Structure.](image)

### CONCLUSION

The purpose of IP address lookup is to find longest matched prefix with the destination address of incoming packet among those prefixes in a routing table. In
this paper, the proposed algorithm consists of two parts, which are multi-bit trie and B-tree. The high 16-bit of IPv6 in the routing table are relatively fixed and this part is processed. In the first part, we use the multi-bit trie to store this part has the advantage of reducing the search range and improving the retrieval speed. In the second part, we use the structure of B-tree and it can reduce the depth of a tree.

REFERENCES