Overlapping Hash Trie: A Longest Pprefix First Search Scheme for IPv4/IPv6 Lookup

Qiong Sun, Zhenqi_x Li, Yan Ma

1. Network Information Center 2. School of Computer Science and Technology
Beijing University of Posts and Telecommunications, Beijing, 100876, P.R.China

Abstract—A router needs to perform a longest prefix matching on address lookup while most of algorithms so far are shortest prefix first search scheme except for LPFST(Longest Prefix First Search Tree)[1]. In this paper, we propose another longest prefix first search scheme of overlapping hash trie (OHT) for IPv4 and IPv6, which resolves the common problem facing at all longest prefix first search schemes including LPFST. We also optimize OHT by using modified tree bitmap[2] to reduce memory cost further more. The results show its average search memory access number for IPv4 is only 23% of LPFST and optimized IPv6 scheme is 14% of LPFST. It also scales well in update and storage.

I. INTRODUCTION

The explosive growth in Internet has brought great challenge to routers. Generally, a router not only have to support wire-speed performance, but also have to handle frequent updating (more than 1000 per second[3]). Moreover, since backbone routers normally have tens of thousands routing entries, the algorithm must have the ability to handle large routing tables and have low storage cost. And besides, we also have to get well prepared for the 128-bit algorithm of next generation internet protocol -- IPv6. Therefore, there are four main requirements for a good IP lookup scheme: search time, update time, storage and scalability.

With the appearance of CIDR (Classless Inter-Domain Routing), IP lookup can be modeled as longest prefix matching (LPM). However, most algorithms search shorter prefixes first[2,4-10] except for LPFST[1] and the lookup process can not stop until to the matched-longest prefix. Although longest prefix first search (LPFS) scheme can stop immediately when meeting a match, the bottleneck that choosing the entry in numerous long-prefixes within a few memory accesses is still unsettled and so the increased performance from LPFS is very limited compared to SPFS(shortest prefix first search) scheme.

In this paper, we develop a novel longest prefix first search scheme of hash overlapping trie. It can not only find out the longest matching prefix within a great many long-prefixes immediately taking advantage of hash, but can also support fast update making use of original trie without pre-computation and value-based segment (some early range-based segments[10] and leaf-pushing[7] related segments can not support fast update). What’s more, we overlap hash on the sub-tries, and thus the sub-trie height can be cut to only 8 in IPv4 and 16 in IPv6 (optimized scheme) without memory expansion or backtracking. Further more, tree bitmap[2] is also introduced to our OHT which will greatly reduce memory cost and make it easy to implement on hardware. In performance evaluation, we can see that our average lookup performance is only 4.2 hash accesses in IPv4 and 8.6 in IPv6, which is only 23% of IPv4 scheme of LPFST and 14% of IPv6, as well as update memory accesses.

The rest of paper is organized as follows: Section 2 reviews previous work of LPM. Section 3 proposes OHT of IPv4 and IPv6. In section 4, we introduce tree bitmap to our algorithm and optimize our IP6 scheme. Section 5 presents the performance evaluation of the proposed scheme, together with the comparison LPFST. Finally, a conclusion is drawn in section 6.

II. PREVIOUS RELATED WORK

The IP lookup algorithms proposed so far can be broadly divided into trie-based, range-based and hash-based and they can implement on software or hardware or both. Hardware solution like TCAM (Ternary Content Addressable Memories) can search the contents in parallel while software solutions can benefit from low cost, flexible and scalability. Here, we mainly discuss about trie-based and hash-based algorithms which related to our algorithm.

A. trie-based algorithm

The original binary trie organize the prefixes with the bits of prefixes to direct the branching. However, since it’s worst case memory access number can be so high that many other techniques have been integrated to reduce the height of trie
(like Patricia[4], multibit trie[5], LC trie[6], etc), but the usage of these techniques often cause hard-updating or memory expansion and some of them[4,6] are relatively difficulty to implement on hardware. Besides, there are also some compression techniques like bitmap[7] which can greatly reduce memory cost and be easy to implement on hardware, but these techniques do not support incremental update. Tree bitmap[2] firstly improved it by using two bitmaps: internal bitmap and external bitmap. The internal bitmap identifies the true prefix in multibit-node while the external bitmap identifies the existing child-node corresponding to this multibit-node. Thus, it successfully get rid of leaf-pushing and made it easy to update.

The latest longest prefix first search tree(LPFST[1]) is also a trie-based algorithm and the tree is constructed with longer prefixes on the upper level and shorter prefixes with the lower level. When there are too many prefixes of same length to store in just one level, they can expand to lower levels with certain bits directing the branch. Although the searching procedure can stop immediately when meeting a match, the common problem mentioned in section one is still unresolved. As a tree by its nature has a contradiction that the upper level has less nodes while the longer prefixes in a routing table have larger number. As a result, LPFST has to store over several levels for same-length prefixes(say 8 levels for all 24-length prefixes), which result in high memory access number for searching and updating.

B. hash-based algorithm

Hash has an outstanding feature of nearly O(1) run-time and is a wildly used technique in IP lookup[8–10], packet classification, etc. Many network processors have also build-in hash unit in it. Waldvogel in [10] proposed a binary search on length-based hash tables which can have the worst case memory accesses of O(logW). Nevertheless, it includes a lot of pre-computation which leads to difficulty-update.

III. OHT Of IPv4/IPv6

A. hash-based algorithm

Assuming that the length of IP address is 8 and the routing table is depicted as Fig. 1.

In OHT, we mainly use three data structures: segment, sub-trie and hash table and three steps to construct OHT

![Diagram](image)

Figure 1. The ordinary trie based on the routing table defined in it

(Fig 2). We can notice in Fig. 1 that there is no prefix shorter than 2 in the routing table, so a value-based segment is firstly constructed at length 2 (SL for segment length). Secondly, we carve out a two stride sub-trie between length 2(TSL for trie_start_length) and 3 (TEL for trie_end_length) and overlap the next two-stride prefixes of length 5 and 6 on its own sub-trie. As it is a longest prefix first search scheme, we overlap the prefixes of length 6 on trie_length(TL) 2 (say prefix G overlap on D) and the ones of length 5 on TL 3 (F overlap on the node under C). We should keep in mind that every TL is overlapped by the long-prefixes with fixed corresponding length (CL), for instance, CL₂ is 6 on TL₂ and CL₃ is 5 on TL₃, and they has the following equation:

\[ CL_a = 2^{*TEL} + 1 - TSL_a \]  \[ (1) \]

Finally, we put prefixes longer than 6(TEL) in hash tables overlapping on segment. As one segment node can have more than one length, we introduce another structure: hash index link to indicate the length of its hash table and they are organized in length-descending order.

In conclusion, we classify all the prefixes into 4 categories based on their lengths and every class has its data structure to deal with (summarized in table1). There are four kinds of nodes in sub-trie and segment: hash index prefix (for longer prefixes), true prefix (for shorter prefix), hash index prefix & true prefix and the dummy ones. Although there are dummy nodes in OHT, their memory cost is still acceptable (discussed in Section 4) and the avoidance of path-compression[4] related techniques leads to fast update.

In OHT, the height of sub-tries is greatly reduced by overlapping hash on them and we can stop searching when meeting a match(say G in hash table) and mark the present matching prefix(say D) at the same time. In this way, we not only reduce the number of long-prefixes by classifying them with the first several bits and find out in O(1) lookup-time, but can also avoid backtracking to shorter ones as well.

<table>
<thead>
<tr>
<th>Prefixes Of Corresponding Data Structure</th>
<th>Data Structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>TSL – TEL</td>
<td>segment</td>
</tr>
<tr>
<td>TEL – 2*TEL + 1 – TSL</td>
<td>segment + sub-trie</td>
</tr>
<tr>
<td>&gt;2*TEL + 1 – TSL</td>
<td>segment + hash index + hash table</td>
</tr>
</tbody>
</table>
B. OHT important parameters

For the routing table in Fig. 1, the parameters like SL, TSL, TEL can be chosen easily, but for real routing table, these parameters should be defined based on the analysis of real routing table. We explore a larger number of real-world IPv4 routing tables and forecast IPv6 routing table based on the current IPv6 Address Allocation and Assignment Policy[11~12]. The conclusion of prefix distribution is summarized in table2.

In accordance, we choose these parameters with the following guidelines.

1. SL should be the length less than which there are no prefixes. Only in this way can a value-based segment be formed without prefix-expansion.

2. CLTSL should be the length greater than which there are few distinct length prefixes, so that the length of hash index link on segment will not be long.

3. The way to search longer prefixes should overlap with shorter ones, thus we can mark the present-matching prefix and avoid backtracking.

The above table3 shows these parameters we choose for IPv4 and IPv6.

C. OHT construction

Initially, all these four data structures are empty and the next-hop is initialized to default one. And then, node x is inserted into different data structures according to its length. The pseudocode of Insert(x, length) is as follows and Get(prefix, a, b) function is to return the value between the a-th and the b-th bits of prefix(from left to right).

Function Insert(x, length)
{ index = Get(x, 0, SL);
  case 1: length < SL
    seg[index]->next_hop = next_hop;
  case 2: TSL<length <= TEL
    change the next-hop of sub-trie node under seg[index];
  case 3: TEL<length <= CLTSL
    insert hash_table of sub-trie node under seg[index];
  case 4: length > CLTSL
    insert segment hash_index and hash_table at seg[index];
}

Function Lookup(x)
{next_hop = default next-hop;
  index = Get(x, 0, SL);
  if seg[index] != NIL then return next_hop;
  else if seg[index]->hash_index == NIL then
    while hash_index == NIL do
      p = Get(x, 0, hash_index->length);
      if hash(p) != h then return next_hop; else hash_index++;;
    else
      node = seg[index]->root; level = TSL;
      while node != NIL do
        if hash(p) != h then node = lchild(node);
        else node = rchild(node);
        if hash(p) != h then return next_hop;
        if node->np == default hop then next Hop = node->np;
        level++;
      return next_hop;
}

D. IP Lookup in OHT

When a packet arrives, for example 1000101, it first goes to 3rd segment node (as in Fig. 1 routing table) since its first 2(SL) bits is 10. As there is no pointer to the hash index, it then goes to the left sub-trie as the 3rd bit of the prefix is 0. Here, we first goes to the hash table pointed by D and find the matching G. This is the longest matching prefix we are looking for, so the searching can just stop. Fig. 4 depicts the lookup function.

E. OHT updating operation

Update operation of OHT is very simple. When inserting an entry, we should just execute the insert algorithm in construction while for deleting or modifying, we could execute the similar algorithm of insert(x, length) to find the node and change the next hop information either to a default one or to the modified one. It is quite simple and flexible.

IV. OPTIMIZATION WITH TREE BITMAP

A. OHT using tree bitmap

Every sub-trie in OHT can be regarded as several multibit-nodes in tree bitmap, so we can adopt these bitmaps to reduce memory cost. However, as in OHT there is another index to hash table, we should introduce another hash bitmap besides these two bitmaps. Thus, the set bit in internal bitmap can either be a true entry, a hash index entry or both of these two cases. Fig 6 depicts a sub-trie bitmap of Fig. 1.
The lookup process in the sub-trees with bitmap is almost the same with tree bitmap except that we first compare with hash bitmap to find if there is a matching in hash table. For software solution, we can choose the subtree based on cache line size (like 4 for 8K cache line size).

B. OHT IPv6 Optimization

For current 32 word size machine, a single 128-bit IPv6 prefix needs four memory accesses. As a result, both the height of trie and the length of prefix will increase by 4 times and hence the performance of IPv6 algorithm will decrease greatly compared to IPv4 scheme in practice. We can notice that in IPv6 routing table, there will be very few entries between the length of 65 to 128. As a result, we can just use two modified IPv4 searching parts for IPv6: one for the first 32 bits and the other for the second 32 bits, adding some special operations for prefixes of length between 65 to 128. In each part, either the length of prefix or the trie height is limited to one word size and so the average performance is approximately twice of IPv4 scheme, by which way we can get rid of multiplying these two factors. This can be regarded as a framework for upgrading to IPv6.

With these considerations, we optimize our OHT in IPv6 to have two parts. In part one, the segment is overlapped by the prefixes of length 16 to 32 together with the length between 65 and 128. While in part two, as there is no length between 33 to 64 with no prefix, only sub-tree can be constructed with 33 for TSL and 48 for TEL. These prefixes in part 2 use the first 32 bits as indexes storing in part 1 hash table. Thus, the height of sub-trie in IPv6 is only 16.

V. PERFORMANCE EVALUATION

Usually, a trie-based algorithm can be evaluated by its height, the average tree height (in table5), the average lookup and update (10000 entries) memory access number (hash access in our algorithm), as well as the memory requirement (KB). Due to paper limitation, we only show the result of routing table AS4637(02/01/06) and AS1221(02/01/06) with LPFST and OHT(whith or without bitmap).

We can see from table4 that OHT for IPv4 has much better performance than LPFST. As they are both longest prefix first search schemes, their average lookup access can be less than average height. But as LPFST does not resolve the common problem of longest prefix first search scheme, its enhanced performance is very limited, while in our algorithm it gains more than one average access.

![Figure 6. The height(av) (left) and lookup(av) (right) for IPv6.](image)

The height has a range due to the size of hash table and we also estimate that our dummy nodes in sub-trees only cost 152K for AS4637 routing table. From Fig. 6 we can see that although the average height of LPFST for IPv6 is about twice of OHT for IPv6, its average lookup access number increases by another 2 times as every node in the tree has much longer length than before. On the contrary, as the length of prefix in each part of our IPv6 optimized OHT is limited to one word size, there is no additional factor to multiply. This result to only 14% memory accesses of LPFST IPv6 scheme.

VI. CONCLUSION

In our algorithm, we introduce a longest prefix first search algorithm OHT for IPv4/IPv6 which can resolve the general problem facing at all LPFS schemes. Besides, modified tree bitmap is also introduced to reduce the memory cost further more. It has much better average lookup and update performance than LPFST and the result for IPv6 optimized scheme is also very promising.

<table>
<thead>
<tr>
<th>TABLE IV. PERFORMANCE COMPARISON OF IPv4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td>AS4637(158800entries) OHT</td>
</tr>
<tr>
<td>AS1221(160469entries) LPFST OHT</td>
</tr>
<tr>
<td>Height 24 8 24 8</td>
</tr>
<tr>
<td>Height(av) 18.78 5.79 18.85 5.82</td>
</tr>
<tr>
<td>Lookup(av) 18.69 4.06 18.80 4.21</td>
</tr>
<tr>
<td>Update(av) 136.064 33.923 138.086 35.652</td>
</tr>
<tr>
<td>M/KB of bitmaps 1411 850-1554 619-1308 628-1326</td>
</tr>
</tbody>
</table>

REFERENCES