High-speed IP address lookup using balanced multi-way trees

Hyesook Lim a,*, Wonjung Kim a, Bomi Lee b, Changhoon Yim c

a Department of Information Electronics, Ewha Womans University, Seoul, Republic of Korea
b Digital Media Research Center, MCT Group, LG Electronics, Seoul, Republic of Korea
c Department of Internet and Multimedia, Konkuk University, Seoul, Republic of Korea

Received 6 April 2005; received in revised form 19 October 2005; accepted 6 December 2005
Available online 5 January 2006

Abstract

Rapid growth of the Internet traffic requires more bandwidth and high-speed packet processing in the Internet routers. As one of the major packet processing performed in routers, address lookup determines an output port using the destination IP address of incoming packets. Since routers should perform address lookups in real-time for hundred millions of incoming packets per second referring a huge routing table, address lookup is one of the most challenging operations. In this paper, we propose a multi-way search architecture for IP address lookup which shows very good performance in search speed. The performance evaluation results show that the proposed scheme requires a single 282 kbyte SRAM to store about 40,000 routing entries, and an address lookup is achieved by 5.9 memory accesses in average.

© 2005 Elsevier B.V. All rights reserved.

Keywords: IP address lookup; Balanced tree; Multi-way tree; Router; Memory access; Binary prefix tree

1. Introduction

Due to the increasing number of domains and hosts connected to the Internet, packet forwarding in the Internet routers is more and more complicated. As one of the major operations in packet forwarding, IP address lookup should be performed in real-time for each incoming packet, and it becomes the bottleneck of router performance. A lot of algorithms and architectures have been studied for efficient IP address lookup. Their performance is evaluated using several metrics such as number of memory accesses, required memory size, flexibility in route update, scalability toward larger routing table, and pre-processing requirement. Among them, the number of memory accesses are the most important since it is directly related to lookup speed.

In this paper, we propose a software-based address lookup scheme which shows very good performance on those metrics. In the proposed scheme, a routing table is divided into multiple balanced trees stored into a memory, and multi-way search is performed in each tree. The rest of the paper is organized as follows. In Section 2, previous works are briefly summarized. Section 3 presents our proposed architecture. Section 4 shows the simulation results using real database and performance comparison with previous works. Brief conclusions are provided in Section 5.

2. Previous works

Routing table entries have prefixes which represent network parts of IP addresses connected to the Internet. IP address lookup problem is to find the best matching prefix (or the longest matching prefix) among prefixes in the routing table with the destination IP address of input packet. A number of previous IP address lookup schemes are categorized as follows.
First one is a ternary content addressable memory (TCAM)-based scheme [1]. TCAM performs IP address lookups for entire entries concurrently with single memory access cycle. However, it is more expensive than common memory, and it has smaller storage space than a same size SRAM as well as it has higher power consumption. Therefore, it is impractical to implement routing table with several hundred thousands of prefixes using TCAM. Moreover, TCAM has a scalability issue to IPv6.

Second, trie is the most common tree-based data structure which represents a routing table. Trie stores a prefix into a node which is defined by the path from the root of the trie. Table 1 shows an example set of prefixes. Using the prefix sample set given in Table 1, Fig. 1 shows the binary trie [2]. In Fig. 1, black nodes represent prefixes and white nodes represent un-assigned internal nodes. Search is performed in bit-by-bit basis, i.e., starting from the root, search moves onto left child or right child depending on the corresponding input bit 0 or 1, respectively. The number of memory accesses is proportional to the length of prefixes in a binary trie, and the number of memory accesses is excessive because of the empty internal nodes as shown in Fig. 1. In order to reduce the number of memory accesses in a binary trie, path-compressed trie removes un-assigned single-child nodes, multi-bit trie [3] inspects more than one bit at a time using prefix expansion [4], and level-compressed trie applies multi-bit trie with path compression technique [5]. In [6], prefix shorter than 24 bits are compressed trie applies multi-bit trie with path compression technique [5]. In Fig. 1, black nodes represent prefixes and white nodes represent un-assigned internal nodes. Search is performed in bit-by-bit basis, i.e., starting from the root, search moves onto left child or right child depending on the corresponding input bit 0 or 1, respectively. The number of memory accesses is proportional to the length of prefixes in a binary trie, and the number of memory accesses is excessive because of the empty internal nodes as shown in Fig. 1. In order to reduce the number of memory accesses in a binary trie, path-compressed trie removes un-assigned single-child nodes, multi-bit trie [3] inspects more than one bit at a time using prefix expansion [4], and level-compressed trie applies multi-bit trie with path compression technique [5].

As one of the most recent tree-based approaches, binary prefix tree (BPT) [12] improved search speed very efficiently by removing empty nodes in trie. In order to construct a tree without empty node and perform binary search, this scheme made several definitions concerning the relationship of prefixes of different lengths as shown in Fig. 2.

Using the same sample set, Fig. 3 shows the binary prefix tree (BPT). While the trie has many empty internal nodes which cause excessive number of memory accesses, BPT includes no internal nodes and hence reduces the number of memory accesses very efficiently. However, for proper binary search, enclosures (prefix a in Fig. 3) in BPT should be located in upper level of tree than the prefixes which have enclosures as a substring, and this limitation causes the constructed BPT highly unbalanced depending on prefix distribution. The required number of memory accesses depends on the depth of the tree, and the tree depth of BPT is also $O(W)$, where $W$ is the maximum length of prefixes. Since BPT is not balanced, each

| Prefix samples | a  | b  | c  | d  | e  | f  | g  | h  | i  | j  | k  | m  | n  | o  | p  | q  | r  | s  | t  | u  | v  | w  | x  | y  | z  |
|---------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|               | 000| 001| 010| 010001101| 0100010101| 010001101| 0110100000| 0111|| 100| 10101| 1011| 11000| 11001| 11010010| 11010011| 11011| 111000110| 11101| 1111| 000000| 00001| 00010000| 000101001| 000110101|

However, 32 Mbytes of memory is required to store $2^{24}$ entries. The scheme proposed in [7] first constructs a forwarding table with $2^{16}$ entries after expanding prefixes into 16 bits, and then builds sub-trees pointed by each entry for prefixes longer than 16. This scheme requires long pre-processing time to construct sub-trees.

There are hash-based schemes. Several schemes have been proposed to apply hashing for IP address lookup [8,9]. Waldvogel et al. [8] proposed to organize a routing table by prefix length and apply a binary search on the prefix length in the routing table. In accessing prefixes in each length, hashing is used. The binary search requires the worst case of $O(\log D)$ memory accesses ($D$ is the number of different levels in trie). However, this scheme requires long pre-processing to compute the best matching prefix of each entry and markers, and hence the routing table update is not trivial. By constructing multiple routing tables and multiple hash functions organized by prefix length, Lim et al. [9] suggested parallel hashing for each prefix length. The number of memory accesses is varied because of the binary search on sub-tables in their scheme.

One of the successful attempts in reducing the required number of memory accesses in trie is Lulea scheme [10]. The scheme reduces a forwarding table into a small data structure which fits into a cache. However, incremental update is not possible in this scheme because of the pre-processing requirement, and it is not appropriate for a large table.

In binary search on range [11], each prefix is represented as a range using the start and the end of the range, and hence the range endpoints for $N$ prefixes partition the space of addresses into $2N + 1$ disjoint intervals. The algorithm uses binary search to find out the interval in which a destination address lies. However, since each interval should correspond to a unique prefix match with the best matching prefix, the algorithm requires pre-computation of this mapping and storing it with range endpoints. Hence this scheme does not provide incremental update.

As one of the most recent tree-based approaches, binary prefix tree (BPT) [12] improved search speed very efficiently by removing empty nodes in trie. In order to construct a tree without empty node and perform binary search, this scheme made several definitions concerning the relationship of prefixes of different lengths as shown in Fig. 2.

Using the same sample set, Fig. 3 shows the binary prefix tree (BPT). While the trie has many empty internal nodes which cause excessive number of memory accesses, BPT includes no internal nodes and hence reduces the number of memory accesses very efficiently. However, for proper binary search, enclosures (prefix a in Fig. 3) in BPT should be located in upper level of tree than the prefixes which have enclosures as a substring, and this limitation causes the constructed BPT highly unbalanced depending on prefix distribution. The required number of memory accesses depends on the depth of the tree, and the tree depth of BPT is also $O(W)$, where $W$ is the maximum length of prefixes. Since BPT is not balanced, each
entry in BPT should include pointers indicating locations of its children, and this consumes fair amount of memory. Because of the limitation in constructing tree, this scheme also does not provide incremental update.

The proposed architecture in this paper is based on the idea that a binary prefix tree composed with disjoint prefixes is perfectly balanced. In other words, if prefixes are sorted to multiple groups so that each group only includes disjoint prefixes, then multiple balanced trees can be constructed. Pure binary search is performed sequentially on those trees. The number of memory accesses is determined by the number of routing entries not the length of prefixes, and hence the proposed scheme is easily extended for IPv6.

In this paper, we propose multiple balanced trees for binary search. In order to well utilize cache line size [11] and improve the search speed, the extended work to multi-way search is also presented. While the binary search requires $O(\log_2 N)$ memory accesses, $k$-way search requires $O(\log_k N)$ memory accesses. A range table is used to further reduce the number of memory accesses in the proposed scheme.

3. The proposed architecture

3.1. Prefix sorting

Prefixes in routing table are primarily sorted using the definition of BPT, and as a result, each prefix has its own level and type. For a given set of sample prefixes in Table 1, sorted result is shown in Fig. 4. For example, 000* is
the first level enclosure, 001* is the first level disjoint, 00001* is the second level disjoint prefix, and so on.

3.2. Multi-way trees

Fig. 5 shows the proposed balanced binary trees. The main tree is constructed with the first level disjoint prefixes and the first level enclosure (prefix a in this example). The sub-tree of the first level enclosure is constructed with the second level disjoint prefixes. Comparing with Fig. 3, two independent trees are constructed and each tree is well balanced. Binary search is performed on the main tree first, and if we match with an enclosure, the search is directed to the sub-tree of the enclosure for a longer match. For all other matches in the main tree, since they are disjoint prefixes, search is completed in the main tree. Since the trees in Fig. 5 are balanced trees, each node does not have to store pointers to its children in this scheme.

Fig. 6 shows the proposed balanced multi-way trees which organize more than one prefixes in each node and compare those prefixes with input at the same time. We have shown an example of 4-way tree in Fig. 6.

3.3. Building routing table

In order to further improve the search performance by confining the search space in the main tree using first eight bits of prefixes, we have a range table. Range table has $2^8$ entries, and each entry has a pointer to the main table which has the prefix starting with the index of the range table entry. Entries of range table additionally have a field indicating the number of main table entries corresponding to its index, and binary search is confined in those entries. Fig. 7 shows a routing table constructed using the proposed binary search scheme. It is shown that 20 prefixes belonged to the main tree are stored into a memory in ascending order, and then prefixes belonged to sub-trees are stored. Each entry has a pointer for output port information (out-port-pointer, not shown in Fig. 7). Enclosure prefixes additionally have a sub-tree pointer pointing its sub-tree location and the number of entries included in the sub-tree. In the example of Fig. 7, the first three bits of prefixes are used as the index of range table, and those bits are not stored into the main table entry, and it is represented using parentheses.

The routing table structure of the multi-way scheme is different from the binary scheme. In k-way search scheme, we proposed that once root node is stored into a memory location, its k child nodes are stored in consecutive memory locations, next $k^2$ child nodes are stored in consecutive memory locations, and so on. The existence of child nodes is indicated by child valid bits. Fig. 8 shows a routing table constructed by proposed 4-way search scheme. Each row stores three prefixes and four child valid bits. The usage of child valid bits is shown in Fig. 9. If there exists a child node which is smaller than Prefix1, the ChildValid1 is set, if there exists
a child between Prefix1 and Prefix2, ChildValid2 is set, and so on. For example, in the memory location 7 in Fig. 8, it is indicated that the Prefix1 has a child smaller than itself by having the ChildValid1 set, and the child node exists in the memory location 8.

### 3.4. Search

In the proposed scheme, search is completed when a matched disjoint prefix is found or no match is found. The out-port-pointer is remembered on each new match.

**Fig. 7. Routing table using proposed binary tree.**

<table>
<thead>
<tr>
<th>Prefix (Length)</th>
<th>Sub-tree Pointer</th>
<th># of Items</th>
</tr>
</thead>
<tbody>
<tr>
<td>a (000)</td>
<td>20</td>
<td>5</td>
</tr>
<tr>
<td>b (001)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>c (010)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>d (011)000110</td>
<td></td>
<td></td>
</tr>
<tr>
<td>e (011)0010101</td>
<td></td>
<td></td>
</tr>
<tr>
<td>f (011)001101</td>
<td></td>
<td></td>
</tr>
<tr>
<td>g (011)0100000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>h (011)1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>i (100)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>j (101)01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>k (101)1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>m (110)00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>n (110)01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>o (110)10010</td>
<td></td>
<td></td>
</tr>
<tr>
<td>p (110)10011</td>
<td></td>
<td></td>
</tr>
<tr>
<td>q (110)101</td>
<td></td>
<td></td>
</tr>
<tr>
<td>r (110)11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>s (111)000110</td>
<td></td>
<td></td>
</tr>
<tr>
<td>t (111)01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>u (111)1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>v (000)000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>w (000)01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>x (000)1000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>y (000)101000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>z (000)110101</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
and returned when search is done. Using the first eight bits of incoming destination address as index of the range table, we first have the search space of the main tree. Binary search or multi-way search is performed to find out a matched prefix in the search space. If the prefix of pointed entry in search is not matched with input, the search is moved to its child node. If we have a matched disjoint, the search is done. If we have a matched enclosure, the search is continued on its sub-tree.

For example in Fig. 8, for the input of 000000111, the first three bits are 000, and hence the search space is address 0 of the main table. As a result of comparison, we have a match with enclosure (000)*, and according to sub-tree pointer, the search should be continued on the sub-tree which is located in memory index 10. In the index 10, because ChildValid1 is set and there is no matched prefix in index 10 and the input is less than the Prefix1 stored in index 10, search moves to the child node which is located in next memory location. Finally, we have a match with (000)000* which is a disjoint prefix, and hence search is done. (000)000* is the longest matched prefix of the input, and its out-port-pointer is returned.

3.5. Update

As mentioned earlier, most of the existing software-based scheme do not provide incremental update, but it is partially possible in our proposed scheme. In our proposed scheme, search space is partitioned into 256 regions using range table, and empty entries can be added in-between each region for prefix insertion. Hence we can limit the range affected by a prefix insertion into a single region. As will be shown in Section 4, the percentage of affected entries by a prefix insertion is 6.7–15% of the total entries in maximum. For a prefix deletion, we reset the entry valid bit of deleted entry, and the deleted entry is used for
comparison in search procedure, but its output port pointer is not remembered.

3.6. Entry structure

Figs. 10 and 11 show the row structure of the main table and the range table for binary scheme and 4-way search scheme, respectively.

4. Performance comparison

Using C language, the performance of the proposed scheme is evaluated. For various sizes of routing tables [13], Table 2 shows the comparison of our proposed scheme with BPT [12] and binary search on range [11] in the required memory size. Since the proposed schemes produce balanced trees and hence do not have to store pointers to children, it is shown that the proposed scheme requires the smallest memories.

Table 3 shows the comparison in the average number of memory accesses and the maximum number of memory accesses. As expected, the proposed multi-way scheme shows the best performance in the number of memory accesses, and the proposed binary search scheme also shows better performance in the average number of memory accesses than others.

Table 4 shows the performance comparison with previous works for 40,000 entry routing table. Lulea scheme [10] shows the best performance in required memory size but the worst performance in the maximum number of memory accesses. The proposed schemes show the better performance than binary search on range or BPT in every aspect.

Table 5 shows the minimum and the maximum number of entries belonged to a single range. In other words, it is shown the minimum and the maximum number of routing entries which can be affected by a single prefix insertion.

5. Conclusion

We have proposed efficient IP address lookup schemes using balanced trees. In the proposed scheme, routing entries are sorted so that each tree is composed of disjoint prefixes, and hence each tree is perfectly balanced. Binary searches or multi-way searches are effectively performed on balanced trees of the proposed scheme. As the required number of memory accesses in the proposed scheme is $O(\log_k N)$ for $k$-way search, it depends on the number of routing entries not on the length of prefixes. Hence the proposed scheme is easily extended for IPv6 which has 128-bit address space. Both of routing tables for IPv4 and IPv6 can be stored into a single
memory if we apply binary scheme for IPv6 and multiway scheme for IPv4. This also makes sense in search speed since IPv4 would have a lot more routing prefixes than IPv6. Range table is proposed to further reduce the number of memory accesses. The proposed scheme is a software-based scheme which does not require special purpose hardware, and hence it is easily implemented with low cost using general purpose microprocessor.

References

Wonjung Kim received the B.S. degree and the M.S. degree at Information Electronics Department from Ewha Womans University, Seoul, Korea, in 2004 and 2006, respectively. Her research interests are system-on-chip design related to network equipments and multimedia networks.

Bomi Lee received the B.S. degree and the M.S. degree at Information Electronics Department from Ewha Womans University, Seoul, Korea, in 2003 and 2005, respectively. She is now with Digital Media Research Lab, LG Electronics, Seoul, Korea. Her research interests are video codec and application development and multimedia networks.

Changhoon Yim received the B.Eng. degree from the Department of Control and Instrumentation, Seoul National University, Korea, in 1986, the M.S. degree in Electrical Engineering from Korea Advanced Institute of Science and Technology, Korea, in 1988, and the Ph.D. degree in Electrical and Computer Engineering from the University of Texas at Austin, USA, in 1996. From 1988 to 1991, he was an engineer working on HDTV at Korean Broadcasting System, Korea. From 1996 to 1999, he was a Member of Technical Staff in HDTV and Multimedia Division, Sarnoff Corporation, NJ, USA. From 1999 to 2000, he worked at Bell Labs, Lucent Technologies, NJ, USA. From 2000 to 2002, he was a Sr. Software Engineer in KLA Tencor Corporation, CA, USA. From 2002 to 2003, he was a Principal Engineer in Digital Media R&D Center, Samsung Electronics, Korea. Since 2003, he has been an assistant professor in Internet and Multimedia Engineering, Konkuk University, Korea. His current research interests include multimedia communication, video compression, image processing, and multimedia network.