Proceedings of the IASTED International Conference
Communication Systems, Networks, and Applications
October 8–10, 2007, Beijing, China
ISBN CD: 978-0-88986-702-4

IMPLEMENTATION OF A PARALLEL-SEARCH TRIE-BASED SCHEME FOR FAST IP LOOKUP

Roberto Rojas-Cessa,* Laksmi Ramesh, Ziqian Dong, Brian D'Alessandro, and Nirwan Ansari
New Jersey Institute of Technology

ABSTRACT
The Internet Protocol (IP) address lookup is required to be resolved fast to keep up with data rate increases. To cope with the increasing number of entries, solutions for IP lookup based on random access memory (RAM), which store prefixes in a trie-based structure, are of interest. In this paper, we propose a flexible and fast trie-based IP-lookup algorithm where parallel searching is performed. This algorithm performs lookup in a maximum of two memory-access times while using a feasible amount of memory.

KEY WORDS
Trie search, parallel search, prefix expansion, hashing, RAM based.

Prefix	Next Hop
01*	21
10*	28
110*	9
1011*	1
0000*	68
01011*	51
00110*	3
10001*	6
100001*	33
10000000*	54

Table 1. Example of a forwarding table.

1. Introduction

Classless inter-domain routing (CIDR) allows Internet routers to store a large number of Internet addresses compactly. While reducing the number of entries in the forwarding table, CIDR increases the complexity of the address-lookup procedure because the longest prefix match is sought rather than the exact prefix match. An efficient IP-lookup algorithm: 1) performs a small number of memory accesses, if not one, for a single lookup, and 2) uses a feasible amount of memory to store the prefix information. Because of long memory-access times and slow advances in improving memory speed, we consider that reducing the number of memory-access times is critical in keeping up with the ever-increasing data link rates. Furthermore, it is required to keep the required memory amount low for an algorithm to be practical.

A known alternative is a trie-based scheme that uses random access memory (RAM). In the basic trie-based scheme, a binary tree represents all combinations existing in the forwarding table. In this scheme, the worst-case time takes up to 32 memory-access times to find the longest prefix match for IPv4, as described in PATRICIA trees [1]. Other improved schemes are presented in [2], which uses small forwarding tables at the expense of requiring up to 12 memory-access times, in [3], which uses 4-bit strides, requiring up to 8 memory-access times, and in [4], using small memory and up to 3 memory-access times.

In this paper, we present a trie-based IP-lookup scheme, which performs parallel search of the matching longest prefix. To reduce the search complexity, this scheme uses controlled prefix expansion [5]. Our scheme uses independent memories for allowing parallel access, and finds the longest prefix match in a maximum of two memory-access times. The presented algorithm is flexible for routing tables with diverse prefix length distributions. Because memories are separate per prefix length this scheme presents high scalability.

The remainder of the paper is organized as follows. Section 2 describes the data structures used and the components of the proposed scheme. Section 3 describes the lookup procedure of our scheme. Section 4 describes the implementation of this scheme. Section 5 discusses the proposed scheme. Section 6 presents our conclusions.

2. Parallel-Search Trie-based Scheme

The proposed scheme performs parallel access to independent memory blocks, where each block stores the entries existing for each group of prefix length. In this paper, we refer to a prefix length as a tree level, i.e., there are up to 32 levels for IPv4 prefixes. Table 1 shows an example of the contents of a forwarding table using CIDR. Figure 1 shows the CIDR entries of the example table presented in a binary-tree structure. In this case, the tree has eight levels, where level one is indicated by the first node below the root, and level eight at the bottom. Our scheme considers that each level can be searched in parallel. To decrease the number of parallel searches, the number of levels (with prefixes) is minimized, into a small number of target levels. To minimize the number of levels, we use controlled prefix expansion [5]. The target levels can be selected by using the most populated levels of an actual forwarding table while considering the memory amount required by each level. Once the levels are selected, the existing prefixes in the removed levels are expanded to the immediate-longer target level. Figure 2 shows the our previous example with expanded prefixes to levels 2, 5, and 8.

Using the contents of actual routing tables [6], we found that a large number of the prefixes are found between

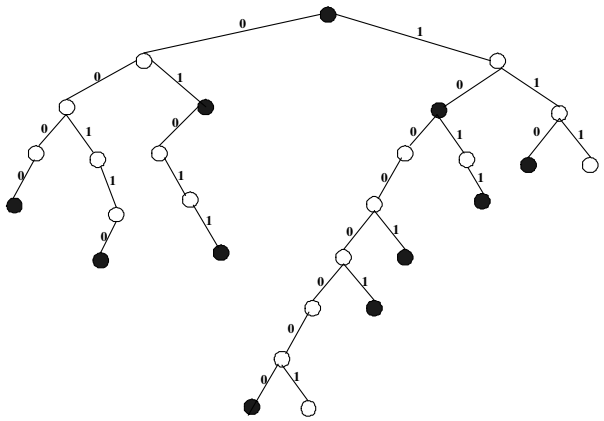

Figure 1. Binary tree representing the forwarding table.

levels 16 and 24. Considering that population, we selected tree levels (or prefix lengths) 8, 16, 24, and 32 as the target levels in our scheme.

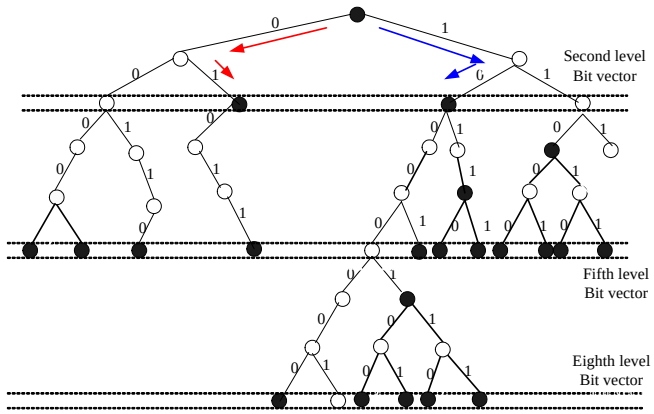

Figure 2. Bit vectors and stored prefixes in extended-prefix tree.

2.1 Data Structures at Target Tree Levels

The set of all possible nodes at each level are represented by bitmaps, where each bit position represents a binary combination corresponding to the bits indicated by the prefix length. In a bitmap, a bit with value of 1 indicates the presence of a stored prefix, and a 0 denotes the absence of it. The left-most bit of the bitmap corresponds to the decimal 0, and the the right-most bit corresponds to the decimal 2^{level}-1, where $level$ is the level number. The bitmaps of levels 8 and 16 are called bit vectors as 2^8 and 2^{16} bits are used, respectively, independently of the existence of prefixes for each bit. The bitmaps for levels 24 and 32 are called bit segments as only partial bit vectors containing one or more prefixes are used.

The **level-8 bit vector** includes the prefixes between expanded levels 1 to 7 and prefixes at level 8. This bit vector, called *prefixval8* is stored in a memory block together with the next-hop information. The **level-16 bit vector** is similar to *prefixval8* but this one includes those prefixes between levels 17 to 24. At this level, there are two other bitmaps: *childval24*, which indicates whether there is one or more prefixes of length between 17 and 24 indexed by *prefixval16*, and *childval32*, which indicates whether there is one or more prefixes with a length between levels 25 and 32 indexed by *prefixval16*. Furthermore, the level-16 bit vectors are physically divided into 32-bit chunks. For every 32-bit chunk, there is an offset value. Therefore, *offsetval16* is the offset value for *prefixval16*, *offsetval24* is the offset value for *childval24*, and *offsetval32* is the offset value for *childval32*. The offset value of bit-chunk of bit n_{16}, where n_{16} is the bit at level 16 in *prefixval16*, *childval24*, or *childval32*, stores the total number of ones accumulated from all previous chunks. The size of these three offset fields is 16 bits each. The **level-24 bit segment**, or *prefixval24* has 256-bit intervals of rooted by *prefix16*, or prefixes at level 24. These intervals are stored in a pseudo-continuous for memory efficiency. This bit segment is denoted as *prefixval24*. The sum of *offsetval24* and the number of ones to the left the *childval24* bit in the chunk is used to find the corresponding interval at level 24. The **level-32 bit segment**, or *prefixval32*, carries those 2^{16}-bit intervals at level 32, which correspond to the subtrees rooted by *prefix16*, with prefixes at level 32, and it is used in the same way as level-24 bit segment. The **next-hop information** for prefixes in each level is stored in several tables, one table per level, called $tablenextY$, where $Y = \{16, 24, 32\}$.

3. Search Procedure

Consider the destination address x of a packet in transit, which can be represented in binary as $x_{31}, ..., x_0$, where x_{31} is the most significant bit. During the first memory-access time, the following fields are accessed: *prefixval16*, *childval24*, *childval32*, *offsetval16*, *offsetval24*, and *offsetval32* with bits $x_{31}, \ldots, x_{16}$. In addition, *prefixval8* and *tablenext8* are accessed, however, with bits $x_{31}, \ldots, x_{24}$.

During the second memory-access time, the following fields are accessed: *prefixval24*, using $x_{23}, \ldots, x_{16}$, of the interval indicated by the value stored in *offsetval24* plus the number of ones on the left of bit *childval24* in the bit chunk. The same is done for *prefixval32*, using *childval32* and *offsetval32*. At the same time, *tablenext16*, *tablenext24*, and *tablenext32* are accessed.

During the second memory-access time, the combined results of the fields *prefixval8, prefixval16, childval24*, and *childval32* are considered to determine which level has a possible matching prefix, or candidate levels. Note that a match at level 16 is confirmed after the first memory-access time. The retrieved next-hop values from those *tablenextY* that are considered candidates are kept.

After the second memory-access time, the next-hop information of the longest prefix value is selected according to the result of *prefixvalueY*, where $Y = \{8, 16, 24, 32\}$.

4. Implementation

The bit vectors and bit segments are stored in a memory block per level. Figure 3 shows the memory blocks for each bitmap and next-hop tables. In the remainder of this paper, we assume that tables, bit vectors, and bit segments

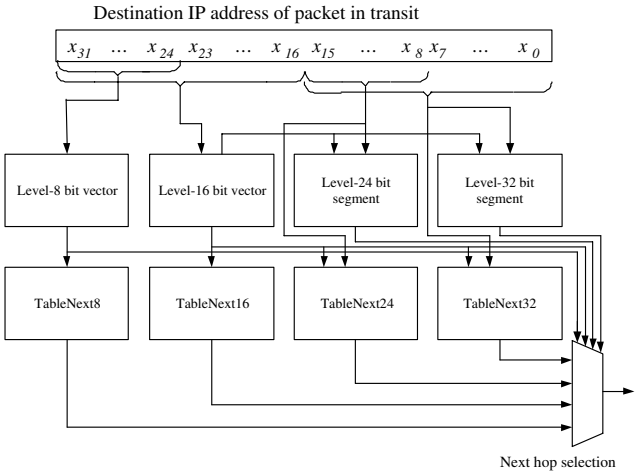

Figure 3. Memory per each target level for parallel search.

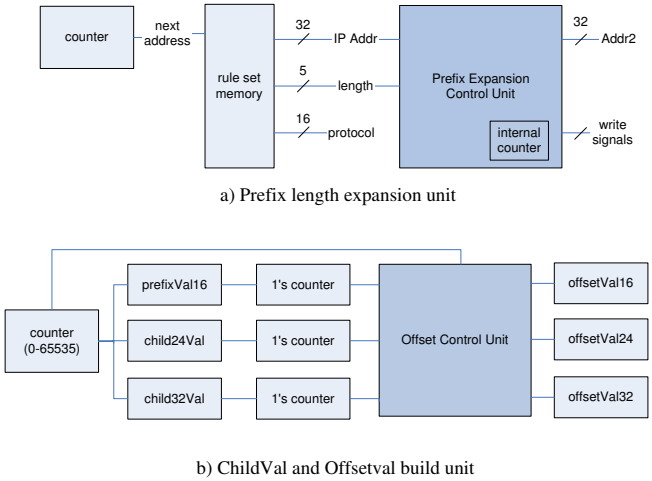

Figure 4. Prefix expansion units for building trie structure.

are stored in separate memory blocks. Specialized hardware is used to build the trie data structure in the memory blocks and to perform the lookup procedure. Each bitmap is build using a prefix expansion unit, as show in Figure 4.a. This unit is used for each used levels. After a prefix is expanded, memory is accessed to store the prefixvalY, childvalY, and offsetvalY, as needed. Figure 4.b shows the unit to calculate offsetvalY for bit vector 16. Figure 5 shows the implementation for the search process for level 24 and 32, as these are the one that need several values from memory to locate their prefixval locations. This unit uses the childval and offsetval values stored at level 16.

5. Lookup Time

Matchings at level 8 are resolved in a single memory-access time, and matching at levels 16, 24, and 32 are resolved in two memory-access times. This memory-access time is obtained at the expense of having memory separately allocated for each level.

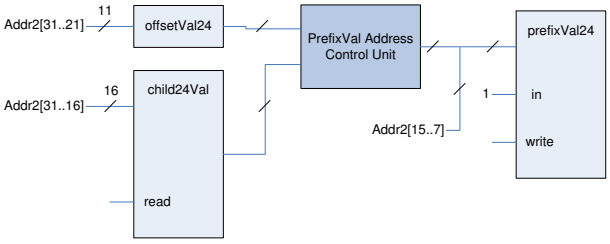

Figure 5. Unit to search for prefixval24. Similar unit is used for prefixval32.

6. Conclusion

We proposed a trie-based IP lookup algorithm that performs parallel search for the longest prefix. Controlled prefix expansion is used to reduce the number of different prefix lengths or levels, and separate memory blocks to reduce the number of memory-access times. As a result, the proposed scheme finds the longest match in up to two memory-access times. As an example, we selected four prefix levels (prefix lengths): 8, 16, 24, and 32. The algorithm searches for a match in levels 16 and 8 at the first memory access. Then it verifies a match in levels 24 and 32, and retrieves all possible next hops, one per level in the second memory access. If matches are achieved at different levels, the match belonging to the longest prefix is selected. This results in having two-memory access time in any case. We also presented the design of special hardware for building the trie structure and lookup procedure. The design gives the option for a fast implementation in addition to using a network processor.

Acknowledgement

This work has been partially sponsored by National Science Foundationtion under Award 0425350.

References

[1] D. R. Morrison, “PATRICIA - Practical Algorithm to Retrieve Information Coded In Alphanumeric,” Journal of the ACM, 15(4), pp. 514-534, October 1968.

[2] M. Degermark, A. Brodnik, S. Carlsson, and S. Pink, “Small forwarding tables for fast routing lookups,” ACM SIGCOMM, pp.3-14, 1997.

[3] D. E. Taylor, J. S. Turner, J. W. Lockwood, T. S. Sproull, and D. B. Parlour, “Scalable IP Lookup for Internet Routers,” IEEE J. of Select. Areas in Commun., Vol. 21, Issue 4 , pp. 522-534, May 2003.

[4] N-F. Huang, S-M. Zhao, J-Y. Pan, and C-A. Su, “A Fast IP Routing Lookup Scheme for Gigabit Switching Routers,” IEEE INFOCOM ’99, Vol. 3, pp. 1429-1436, March 1999.

[5] V. Srinivasan and G. Varghese, “Faster IP lookups using controlled prefix expansion,” ACM Trans. Comput. Syst., pp. 1-40, Feb. 1999.

[6] BGP Table Data, http://bgp.potaroo.net.