Available online at www.sciencedirect.com
SCIENCE DIRECT
Computer Communications 29 (2006) 1927–1935
computer communications
www.elsevier.com/locate/comcom

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.

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.

☆ This research was supported by the Ministry of Information and Communications, Korea, under HNRC-ITRC support program supervised by IITA.
* Corresponding author. Tel.: +82 2 3277 3403; fax: +82 2 3277 3494.
E-mail address: hlim@ewha.ac.kr (H. Lim).

0140-3664/$ - see front matter © 2005 Elsevier B.V. All rights reserved.
doi:10.1016/j.comcom.2005.12.002

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 unassigned 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 expanded to 24 bits, and then initial lookup is performed on the 2^{24} entry table. If there exist longer prefixes, an additional lookup is executed at the second memory using the pointer indicated at the indexed entry of the first memory.

Table 1
Prefix samples

a	000
b	001
c	010
d	0110001101
e	0110010101
f	011001101
g	0110100000
h	0111
i	100
j	10101
k	1011
m	11000
n	11001
o	11010010
p	11010011
q	110101
r	11011
s	111000110
t	11101
u	1111
v	000000
w	00001
x	0001000
y	000101000
z	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 $O(\log_2 D)$ 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 trival. 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