Fast IP Lookups Using Binomial Spanning Trees

Yeim-Kuan Chang

Department of Computer Science and Information Engineering
National Cheng Kung University
Tainan, Taiwan R.O.C.
ykchang@mail.ncku.edu.tw

Abstract—**High performance Internet routers require an efficient IP lookup algorithm to forward millions of packets per second. Various data structures based on binary trie are normally used in software-based IP router design, including network processor-based routers. Binary trie based lookup algorithms have not only simple and easy IP address search process but also the routing entry update process. In this paper, we propose a new IP lookup algorithm based on binomial spanning tree. The proposed algorithm has the same advantages of simple search and update processes as the binary trie-based algorithms. However, the performance of the proposed algorithm is better than the schemes based on binary tries, such as path-compression and level-compression.**

Keywords — **Binary trie, binomial spanning tree, and IP lookup.**

I. INTRODUCTION

The increase of the Internet traffic continues in an unprecedented rate mostly due to the advent of the World Wide Web (WWW) [5]. Backbone routers with link speed of gigabits per second (e.g., OC-192, 10 Gigabits and OC-768, 40 Gigabits) are thus commonly deployed. Among all the fundamental functions of the routers, IP address lookup is the most critical one. Fast lookup algorithms make packet forwarding rate of the routers keep up with the link speed and router bandwidth. These backbone routers have to forward millions of packets per second at each port. In this paper, we focus on the choice of data structure and its adaptation to the typical routing tables with a large number of routing prefixes. We evaluate the performance of the proposed algorithm and other existing ones using a software implementation.

The routing table in a router that is used to lookup an IP address stores an array of entries, each consisting of a network address that is the prefix of a group of IP addresses and the corresponding next port number to the network. When a router receives a packet, it must determine the next port number through which the packet must be forwarded. The longest prefix in the routing table that matches the destination IP address of the packet is the best match prefix (BMP). Sequential search for the BMP has a time complexity of O(N) which is not scalable.

A large variety of routing lookup algorithms was classified and their worst-case complexities of lookup latency, update time, and storage usage were compared in [1]. Among them, a category of algorithms is based on a trie/tree structure. The binary trie is in fact a binary search tree using the bit value (0 or 1) to guide the search moving toward the left or the right part of the tree. The binary tree structure is usually implemented using linked list data structure. Each trie node has the left and right pointers pointing to its left and right sub-tree, respectively. A space efficient array implementation of trie-based algorithms is also possible [12].

Among all the IP lookup algorithms proposed in the literature, only the binary range search proposed in [3, 4] can store the lookup data structure in a sequential array. Instead of trying to store the complete prefixes, the binary range search encodes the prefixes by the start and end addresses of the ranges covered by them. All the start and end addresses of ranges are sorted and stored in a sequential array. The binary search method can then be applied using the array index. Obviously, a subtle design (e.g., “>” and “=” ports) must be employed to make the binary search on the sequential array work. The primary idea of the binary range search is to pre-compute the port number when the target IP is equal to one of the start and end addresses of ranges or locates between two consecutive addresses.

Based upon this primitive trie structure, a set of prefix compression and transformation techniques are used to either make the whole data structure small enough to fit in a cache [9], or to transform the set of original prefixes to a different one in order to speed up the tree traversal procedure [10]. The hardware based lookup algorithms using multi-bit trie proposed in [11] is in fact a variation of the prefix transformation techniques. The extreme case is a 32-bit extended trie which trades a memory consumption of 32 Gbytes and inefficient prefix updates for only one memory lookup latency. We can classify the 32-bit extended trie as a perfect hashing approach which is obviously not minimal. Since finding a minimal perfect hashing table for the whole set of prefixes is difficult, a binary search on prefix lengths is proposed in [7]. In this scheme, a binary search scheme is conducted on a set of hash tables, where prefixes with same length are organized in one hash table. In [8], the authors use CPU caching hardware to perform routing table caching and lookup directly by carefully mapping IP addresses to virtual addresses.

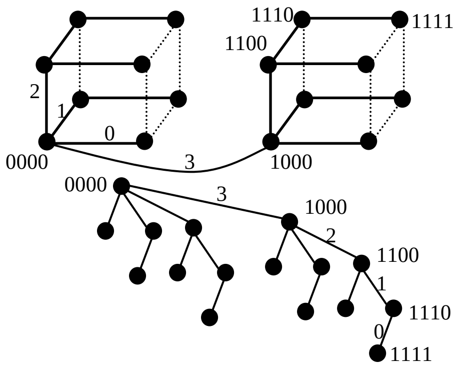

Figure 1: A 4-cube and its corresponding binomial spanning tree.

In this paper, we shall propose a new IP lookup algorithm that uses a binomial spanning tree. The spanning tree is constructed from a hypercube structure of dimension *n* onto which the n-bit prefixes of the routing table are mapped, where *n* is 32 for IPv4 or 128 for IPv6.

Formally, a hypercube or an n-cube consists of 2^n nodes, can be topologically represented as an n-dimensional cube in which a node is located on each of the 2^n vertices of the cube. Each of the 2^n nodes in an n-cube has a unique n-bit binary address, and two nodes are adjacent and connected by a link if and only if their addresses differ in exactly one bit. Subcubes of an n-cube are denoted by ternary strings in {0, 1, *}, where * is the Don't Care bits which can be replaced by either 0's or 1's. For example, 00** is a subcube of dimension 2 in a 4-cube which contains 4 nodes with addresses 0, 1, 2, and 3.

A special tree structure called the binomial spanning tree can be constructed from a hypercube. The fan-out of a node in the binomial spanning tree can be in the range of 0 … *n* in an n-cube. A four dimensional hypercube and the corresponding binomial spanning tree are depicted in Figure 1. In the figure, some of nodes are also shown with their corresponding 4-bit addresses.

The proposed IP lookup algorithm based on the binomial spanning tree has the similar characteristics to that based on the binary tries. In other words, both of the algorithms based on binomial spanning trees and binary tries have the advantages of simple and easy searching mechanism, tree construction, and updates. The basic searching method for the proposed lookup algorithm is also in a bit-by-bit fashion starting from the most significant bit and following the pointers in the nodes. The worst-case number of memory references with the basic binomial spanning tree is less than n since we can easily pick a root node which has the maximum hamming distance of less than n from all other nodes with valid prefixes. We will then develop some techniques to further reduce the number of nodes and the tree depth in the binomial spanning tree using level and path compression and the special properties of the hypercube [10, 12].

The rest of the paper is as follows. Notations, terminology, definitions are introduced in Section 2. In Section 3, we describe the data structure of the proposed IP lookup algorithm using binomial spanning tree. Section 4 describes a software implementation that we use to conduct the experiments for lookup performance comparisons. Finally, a concluding remark is given in the last section.

II. PRELIMINARIES

The notations and terminology used in this paper are first given as follows.

In an n-cube, node *i* has binary address, $(i_{n-1}\, i_{n-1}\ldots i_0)$. The set of node addresses is $\mathcal{N} = \{0, 1, \ldots, 2^n-1\}$, and the set of dimensions is $\mathcal{D} = \{0, 1, \ldots, n-1\}$. The bitwise Exclusive-OR operation is denoted as ^ (used in C language). $|\mathcal{S}|$ denotes the cardinality of a set $\mathcal{S}$.

Definition 1: A binary n-cube is a graph G = (V, E) such that $V = \mathcal{N}$ and $E = \{(i,j) \mid i \wedge j = 2^m$, for all i and j belong to $\mathcal{N}\}$. An edge (i,j) connects nodes i and j through dimension *m*.

Definition 2: The Hamming distance between nodes i and j is $\text{Hamming}(i, j) = \sum_{m=0}^{n-1} (i_m \wedge j_m)$.

Definition 3: A binomial spanning tree (n-BST) with root node $s = (s_{n-1}\, s_{n-2}\ldots s_0)$ is defined [2] as follows.

The set of the root's children is $\{(s_{n-1}s_{n-2}\ldots\overline{s_m}\ldots s_0)\}$ for $m = n-1, \ldots, 0$. The set of children of another node *i* with the address $(i_{n-1}\, i_{n-2}\ldots i_m\ldots i_0)$ is $\{(i_{n-1}\, i_{n-2}\ldots\overline{i_m}\ldots i_0)\}$ for $m = p-1, \ldots, 0$, where $c = i \wedge s$ and $c_k = i_k \wedge s_k$ for $k = 0 \ldots n-1$, and $c_{p-1} = \ldots = c_0 = 0$ and $c_p = 1$. In other words, *p* is the position of the least significant set bit. The sub-BST containing nodes in the p-cube $0..01_p$* is connected to the root node along dimension p.

n3.n2.n1.n0/l/p: the *length format* of prefixes. It represents a prefix of length n associated with a next port number p, where n3.n2.n1.n0 is dotted notation of a 32-bit IP address using 4 octal numbers. The notation n3.n2.n1.n0/l will be used when no confusion is incurred.

$b_{n-1}\ldots b_i{*}\ldots{*}/p$: the *ternary format* of prefixes. It represents a prefix of length n-i associated with a next port number p and b_j = 0 or 1 for $n-1 \geq j \geq i$. When we use $t_{n-1}\ldots t_1t_0$ as the ternary format of a prefix, where t_i = 0, 1, or * (don't care), we must follow the rule that if t_k is * then t_j must also be * for all $j < k$. For simplicity, a single don't care bit is used to denote a series of don't care bits. Thus, the prefix 1* denotes 1**** in a 5-bit address space.

Prefix Enclosure. Consider two prefixes in their ternary format: $A = b_{n-1}\ldots b_i{*}$ and $B = b_{n-1}\ldots b_j{*}$ and assume j > i. Therefore, A is enclosed by B.

Disjoint prefixes. Two prefixes A and B are said to be disjoint if none of them is enclosed by the other.

III. PROPOSED DATA STRUCTURE

Using the binomial spanning tree as the basic structure to store the routing table is not as simple as we first thought. Each vertex of an n-cube is associated with an n-bit binary address which can be directly mapped to a node in the binomial spanning tree. Storing routing table in the binomial spanning tree will be straightforward when all the prefixes of the routing table

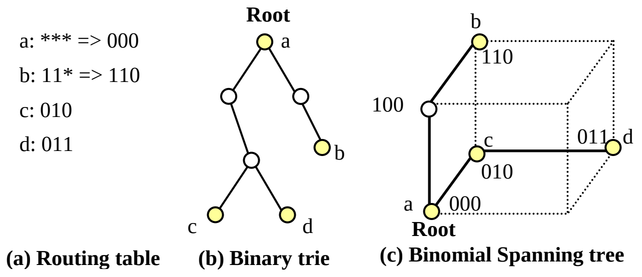

Figure 2: a small routing table example for the proposed algorithm.

are of length 32. However, the prefixes of the routing table are in the range of 1 to 32. We need to devise a method to store a prefix of any length in a node of the binomial spanning tree such that performing an IP lookup will have a correct result. What we do is to convert the prefix $b_{n-1}\ldots b_0/len/p$ to $b_{n-1}\ldots b_{n-len}0\ldots 0/len/p$ and store it in the node with address $b_{n-1}\ldots b_{n-len}0\ldots 0$.

The above conversion leads to a problem that a node may be mapped from more than one prefixes. For example, the node with an address $b_{n-1}\ldots b_0$ can store any one of the prefixes $\{b_{n-1}\ldots b_{n-k+1}1_{n-k}0\ldots 0{*}_x\ldots{*}_0/len \mid len = n - 1 - x \text{ and } x = -1\ldots n - k - 1\}$, called the set of conflicting prefixes in a node. This conflict situation can be solved by two approaches. The first approach uses an additional prefix array to record the conflicting prefixes. The node will contain a pointer to this prefix array. When a lookup ends in a node of the binomial spanning tree containing a non-empty conflicting prefix array, an additional process must be performed to search the prefix array for the proper match. Since this kind of enclosure situation is rare, the LC trie uses this approach [12].

The second approach expands the conflicting prefixes of shorter lengths to ones of longer lengths in such a way that all the expanded prefixes are disjoint. Disjoint prefixes are mapped onto different nodes of the binomial spanning tree. For example, two conflicting prefixes 0*/p and 000*/q in the 4-bit address space are initially mapped to the same node with address 0000. The converted set of prefixes are 01**/p, 001*/p, and 000*/q that are mapped to three distinct nodes with addresses, 0100, 0010, and 0000. One might think that this approach is the same as the approach that is designed to remove the enclosure situations. The following example explains it is not. Consider two prefixes, 0*/p and 01*/q in a 4-bit address space. The former prefix encloses the latter. It can be seen that these two prefixes are not conflicting because they are mapped onto two distinct nodes with addresses 0000 and 0100. This is the approach we adopt in this paper.

Figure 2 shows the binary tree and the corresponding binomial spanning tree for a small routing table. We can see that the depth of the binomial spanning tree is one less than the binary trie and the number of links is two less than that of the binary trie. The number of links is proportional to the required memory space because the links are usually implemented as pointers.

The insertion procedure in the syntax of C programming language for the proposed lookup algorithms is given in Figure 3. This insertion procedure is the building block of the tree construction and update processes. Formally, the insertion procedure is called every time when a prefix is added in the routing table. The last three parameters of the insertion procedure, *ip*, *len*, and *port*, represent the prefix being added. If the hamming distance between the root and the node mapped to the inserted prefix is *h*, there are *h* nodes in the binomial spanning tree that will be traversed or created if necessary in the insertion process, excluding the nodes that will be created by the conflicting node resolution process. Lines 1-9 show the core codes that create the necessary nodes along the path from the root to the final node of the prefix. The input IP is first Exclusive-ORed with the root address. The position of the most significant set bit is then computed by using the 'bsr' (bit scan reverse) instruction of the Intel processor family starting from Intel 80386. When coming to the final node where the input prefix is supposed to locate, we check if a conflicting prefix already exists and perform the appropriate conflict resolution operations. Referred to the line 10 in Figure 3, we assume that if the final node having a port number greater than 0 indicates a conflicting prefix was already assigned to this final node. If no conflicting prefix exists in this final node, we update its *len* and *port* fields.

```
void insertion(node32 *root32, unsigned root_ip,
        unsigned ip,unsigned len,unsigned port)
{
1  unsigned x = root_ip ^ ip, i;
2  while (x != 0){
3    i =BSR(x);//Pentium's bsr instruction(bit scan reverse)
4    if (root32->ptr[i] == NULL)
5         root32->ptr[i] = create_node32();
6    root32 = root32->ptr[i];
7    x = x ^ (1<<i);
8    root_ip = root_ip ^ (1<<i);
9  }/*end while */
10 if (root32->port > 0) {
11      conflict_resolve(root32, root_ip, ip, len, port);
12 } else {
13      root32->len = len;
14      root32->port = port;
15 }
}
```

Figure 3: The insertion procedure for the proposed binomial spanning tree.

Figure 4 shows the procedure for solving the conflicting situation. The prefix assigned to this node is the one with the longest length. The prefix with a shorter length is split into two prefixes that are in turn inserted recursively into the same node.

Finally, we show the lookup procedure in Figure 5. The lookup process works in a bit-by-bit fashion as in the insertion procedure. Each time a node is looked-up when traversing the binomial spanning tree, the matching process is performed. If the input IP matches the prefix stored in the traversed node, the node port number is recorded as default port and the lookup process continues. The final matched node is the best

```
void conflict_resolve (node32 *root32, unsigned root_ip,
        unsigned ip, unsigned len, unsigned port)
{
  unsigned q, lmin, pmin, lmax, pmax;
  if (root32->len == len) return;
  (root32->len < len ? lmin=root32->len : lmax=len);
  root32->len = lmax;
  (root32->len < len ? pmin=root32->port : pmax=port);
  root32->port = pmax;
  q = root_ip^(1<<(31-lmin));
  insertion(root32, root_ip, q, lmin+1, pmin);
  insertion(root32, root_ip, root_ip, lmin+1, pmin);
}
```

Figure 4: The conflict resolution procedure for the proposed binomial spanning tree.

matched prefix whose port number is returned. Otherwise, the default port is returned.

A. Node Representation:

As you may have noticed that the data structure of the node in Figures 3-5 contains fixed number of pointers depending on where the node locates. The root node has *n* pointers since it may have at most *n* children in an *n*-BST. In general, there are 2^k nodes with $n - k - 1$ pointers, for k = 0 to n – 1. A lot of pointer space will be wasted because some of the prefixes may not exist and thus most of pointers are NULLs. We propose to use a bitmap to record which pointers are not NULLs. Take a node with 8 pointers as an example. If only pointers at bits 3 and 5 are not NULL, we use 00101000 as the bitmap and an array of two pointers. When we check if the pointer at bit *i* is NULL or not, we check if the ith bit in the bitmap is set or not. If the ith bit of the bitmap is zero, the corresponding pointer is NULL. If not, we compute how many set bits that precedes the ith bit (inclusive) to locate and follow the pointer to the next-level node.

B. Optimizations:

There are a number of optimization techniques proposed in the literature for the binary trie. Path compression, level compression, and k-level segmentation are examples of the optimization techniques. We will show that these techniques can also be applied to the proposed IP lookup algorithms using binomial spanning tree.

Path compression of the binomial tree is straightforward. The non-prefix node with only one child can be removed by the path compression technique. Figure 2 (c) shows node 100 can be compressed into node 100 which stores prefix b.

Employing level compression technique in the binomial spanning tree is similar to finding a subcube that contains the node at which the level compression is applied. For example, assume there are 14 prefixes that are stored in a binomial spanning tree shown in Figure 6. There exists a 3-cube that contains the root node 0 (prefix 0). The data structure of the root node as shown in Figure 6 contains a modified bitmap, 11***, to indicate that there are nine pointers stored in the node. There are seven pointers pointing the 7 nodes in a 3-cube,

```
unsigned lookup(node32 *root32, unsigned root_ip,
        unsigned traffic_ip, unsigned default_port)
{
  unsigned x = root_ip ^ traffic_ip, i, j;
  while (x != 0){
     if (root32->len != 0) {
        j = 32 - root32->len;
        if ((root_ip >> j) = = (traffic_ip >> j))
           default_port = root32->port;
     }
     i =BSR(x);//Pentium's bsr instruction(bit scan reverse)
     if (root32->ptr[i] == NULL)  return default_port;
     root32 = root32->ptr[i];
     x = x ^ (1<<i);
     root_ip = root_ip ^ (1<<i);
  }/*end while */
  j = 32 - root32->len;
  if ((root_ip >> j) == (traffic_ip >> j))
        return root32->port;
  else return default_port;
}
```

Figure 5: The lookup procedure for the proposed binomial spanning tree.

00***, (nodes 1, 2, 3, 4, 5, 6, and 7) and another two pointes pointing to nodes 8 and 16 along dimensions 3 and 4, respectively. If the incoming IP is 01001, the second pointer (with prefix 8) is first selected because the most significant set bit of IP is in dimension 3. After it reaches at node 8 with address 01000, the same process continues. If the incoming IP is 00101, the seventh pointer will be followed.

The k-level segmentation is mostly used in the hardware-based IP lookup algorithms [1]. For our binomial spanning tree, we also use the k-level segmentation array of 2^k pointers, each pointing to the corresponding sub-binomial spanning tree of dimension n – k. Therefore, when an incoming IP arrives, the most significant k bits are used to locate the corresponding sub spanning tree. Then the usual lookup process can be performed in the sub-spanning tree to find the best matched prefix.

Besides the path compression, level compression, and k-level segmentation, intelligent selection of root node and dual roots, a pair of diagonally opposite nodes in the n-cube, can be used to further reduce the depth of the binomial spanning tree. Consider the example shown in Figure 2 (c). If the node c (010) is selected as root, then the constructed binomial tree rooted at node c has only depth of one. Nodes a, b, and d are all one hop from node c. Therefore, the tree depth can be reduced by carefully selecting a root for a sub-tree.

We know that the hamming distance between the pair of diagonally opposite nodes A and B is n, the maximum, in an n-cube system. Node A is the only node that has a distance n from node B. For example, nodes A and B could be the nodes with addresses 0 and $2^n - 1$. We will use these two addresses for the dual roots as the default ones when we describe our idea as follows.

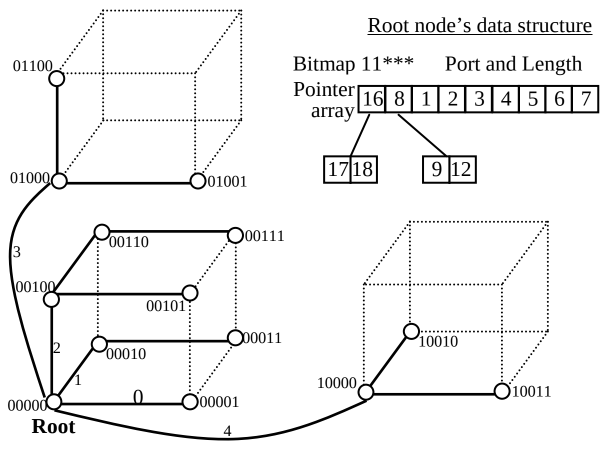

Figure 6: level compression example for the proposed algorithm.

The maximum distance between either A or B and any other node is n/2. Therefore, we can use this property to build two binomial spanning trees, one rooted at A and the other rooted at B. The previous proposed insertion procedure can be employed directly as follows. When the binary address of a prefix using the address conversion scheme described above is closer to root A than B, we insert this prefix in the binomial spanning tree rooted at A. Otherwise this prefix is inserted into the spanning tree rooted at B. After a prefix has been determined to be inserted in the tree rooted at B, the address conversion scheme is different from the original scheme and will be described later. If there exists an address covered by a prefix that is closer to A than B then this prefix is inserted into the tree rooted at A. At the same time, it is possible that there is also an address that is covered by the prefix is closer to B than A. If it is the case then a marker prefix must also be inserted into the binomial tree rooted at B. For example, if the prefix, 3.0.0.0/8, must be inserted both into the trees rooted at A and B. When inserted into the tree rooted at node A, we use the original address conversion scheme. However, when inserted into the tree rooted at node B, we use a different conversion scheme in order to reduce the distance between node B and the converted address. What we do is converting the prefix 3.0.0.0/8 to address 3.255.255.255 by padding ones to the don't-care bits. If the prefix to be inserted is 4.0.0.0/24 then no marker prefix is created.

The lookup process is similar to the insertion. When the incoming IP is closer to root A than B, the lookup process is preformed on the spanning tree rooted at A. Otherwise the lookup process is performed on the tree rooted at B.

IV. Performance Evaluation

In this section, we conduct experiments based on our implementation of the proposed IP lookup scheme. We use two routing tables for the measurements, one is a small table, funet, used in LC paper [12] and another one is a big routing table [6] which reflects the current situation in modern routers. The funet and oix tables contain 41,709 and 120,637 routing entries, respectively. Both tables contain the prefixes of lengths ranging from 8 to 32.

In addition to the proposed scheme, the binary trie and the Level-Compressed (LC) schemes are also considered for comparisons. The binary trie scheme is the most fundamental trie structure. The LC scheme is the optimized version of the binary trie scheme using adaptive internal subtrie expansions, first k-level segmentation, and pre-allocated array. The C codes for the LC trie are obtained from the web site published by the authors. In fact, we also implemented a variant of path compressed scheme. Since the performance of the path compressed scheme does not perform better than LC, its results are not given in this paper.

The methodology of the conducted experiments is taken from the LC paper. The input traffic pattern is taken from the routing table. Therefore, the lookups will always be hits. Notice that the IP of the traffic is randomized before feeding into our simulator. All experiments will be performed with k=0, 8, or 16 level segmentation. The clock cycles for each experiment are measured by using the special instruction, rdtsc (read time stamp counter), provided by Intel Pentium processor. The clock counts obtained from different CPUs may have different scales. For example, the clock counts from Pentium IV are greater than that from Pentium III for the same experiments. It does not necessarily mean that the conducted algorithms perform better on Pentium III than on Pentium IV. We need to convert the clocks to seconds, which is an easy task. However, we will not perform this conversion for the clarity of the figures shown.

In addition, a minor adjustment is made for better performance when selecting a diagonally opposite node of the original root. Since most of the prefixes are of length 24 or shorter, we only consider the first 24 bits when selecting the dual nodes in the schemes with k-level segmentation. For example, consider the i^{th} segment in a 8-level segmentation. The address of the first root is i.0.0.0 as usual. However, the address of the second root is chosen to be i.255.255.0 instead of i.255.255.255. For the proposed scheme without segmentation, the dual roots are 0.0.0.0 and 255.255.255.255.

Figures 7 and 8 show the experiments results of clock cycle counts on a 2.4G Pentium IV processor with 8KB L1 and 256KB L2 caches with a 16-level segmentation. In Figure 7, we can see that there are similar number of peaks for the proposed scheme and the LC. However, the peaks of the proposed scheme move toward to the left end. This means the time taken for the proposed scheme is smaller than the other two schemes. Notice that there are five peaks in LC because the number of levels in LC is five. In Figure 8, we compare the proposed schemes with one root and two dual roots. We can see that the shapes of these two schemes are similar except curves on the right side. There are more hits with longer clock cycles in the scheme with only one root than that with dual roots. This improvement is because lookups with longer cycles in the scheme with one root is transferred to tree rooted at the diagonally opposite root. The lookups with shorter cycles will not be affected.

The results of the same experiments for funet routing table are depicted in Figures 9 and 10. The difference between these two routing tables is not significant. The proposed scheme still performs better than other schemes.

We also conduct experiments on the 1G Pentium III CPU with 16KB L1 and 256KB L2 caches. As shown in Figure 11 without first k-level segmentation, the LC scheme has the same peaks as previous experiments on Pentium IV. The peaks in the curve for the proposed scheme are not as clear as LC scheme. Figures 12 and 13 show the results of LC and the proposed scheme with dual roots with an 8-level and a 16-level segmentation. Figure 14 compares the performance for the proposed scheme with 0, 8, and 16-level segmentation. Obviously, the scheme with a 16-level segmentation performs the best. In order to summarize the performance for all the schemes run on Pentium IV processor, we calculate the average clock cycles and show the results in Table 1. Again, the proposed scheme with dual roots performs better than any other scheme.

V. Conclusions

In this paper, we introduced a new method based on the binomial spanning tree. By mapping the prefixes of different lengths on the vertices of an n-cube, we can construct the binomial spanning tree using the simple tree construction and update procedures. The fundamental binomial spanning tree can be optimized by using the path compression, level compression, k-level segmentation and the property of diagonally opposite nodes. We implemented the proposed scheme and other existing trie-based schemes and showed that the proposed binomial spanning tree based scheme performs the best.

References

[1] M. A. Ruiz-Sanchez, Ernst W. Biersack, and Walid Dabbous, "Survey and taxonomy of IP address lookup algorithms", IEEE Network Magazine, 15(2):8--23, March/April 2001.

[2] S.L. Johnsson and C.-T. Ho "Optimum Broadcasting and Personalized Communication in Hypercubes", IEEE Transactions on Computers Vol. 38, No. 9, Sept. 1989.

[3] M. Akhbarizadeh and M. Nourani, "IP Routing Based on Partitioned Lookup Table," in Proceedings of the IEEE International Conference on Communications (ICC) (New York, NY), pp. 2263-2267, April 2002.

[4] Butler Lampson, Venkatachary Srinivasan and George Varghese, "IP lookups using multiway and multicolumn search", IEEE/ACM Transactions on Networking, Volume 3, Number 3, Pages 324-334, 1999.

[5] Geoff Huston, "Analysis of the Internet's BGP routing table", Internet Protocol Journal, 4(1), March 2001.

[6] David Meyer, "University of Oregon Route Views Archive Project: oix-damp-snapshot-2002-12-01-0000.dat.gz " at http://archive.routeviews.org/

[7] M. Waldvogel, G. Varghese, J. Turner and B. Plattner. "Scalable high-speed IP routing lookups," Proceedings of ACM Sigcomm, pp.25-36, October 1997.

[8] T. Chiueh et al, High Performance IP Routing Table Lookup Using CPU Caching, INFOCOM99'

[9] M. Degermark, A. Brodnik, S. Carlsson, and S. Pink. "Small Forwarding Tables for Fast Routing Lookups." ACM SIGCOMM'97, Palais des Festivals, Cannes, France, pp. 3-14.

[10] K. Sklower, A Tree-based Packet Routing Table for Berkeley Unix, Proc of 1991 Winter Usenix Conf, 1991, pp.93-99

[11] P. Gupta, S. Lin, N. McKeown, Routing Lookups in Hardware at Memory Access Speeds, INFOCOM99'

[12] S. Nilsson and G. Karlsson "IP-Address Lookup Using LC-Tries", IEEE Journal on selected Areas in Communications, 17(6):1083-1092, June 1999.

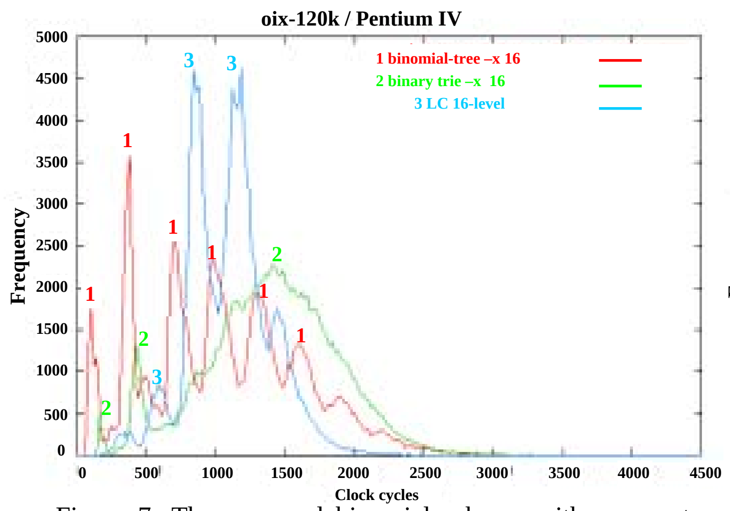

Figure 7: The proposed binomial scheme with one root, binary trie, and LC with a 16-level segmentation for oix routing table on Pentium IV.

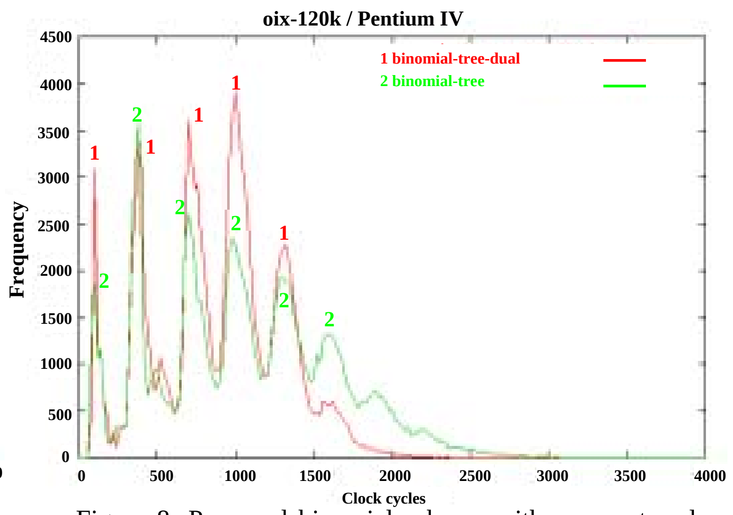

Figure 8: Proposed binomial scheme with one root and that with dual roots without segmentation for oix routing table on Pentium IV.

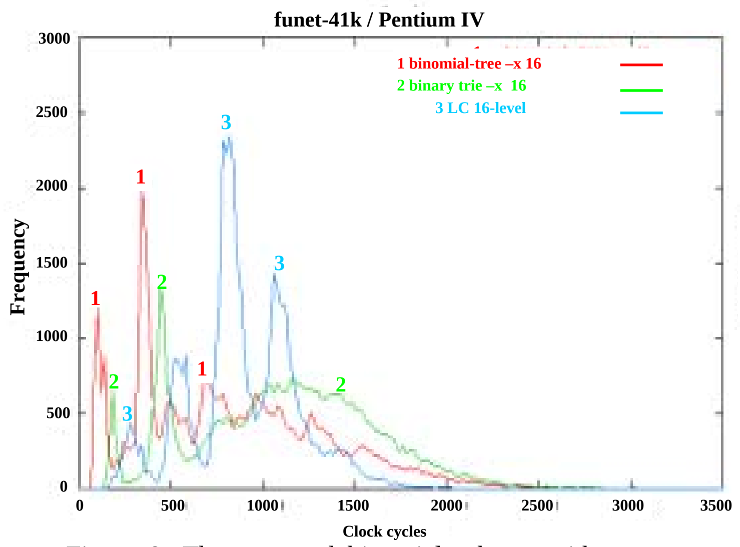

Figure 9: The proposed binomial scheme with one root, binary trie, and LC with a 16-level segmentation for funet routing table on Pentium IV.

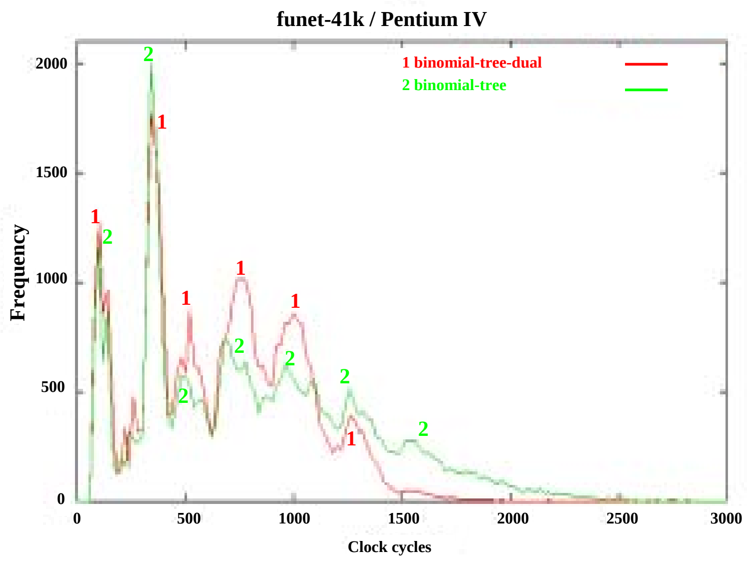

Figure10: The proposed binomial scheme with one root and that with dual roots without segmentation for funet routing table on Pentium IV.

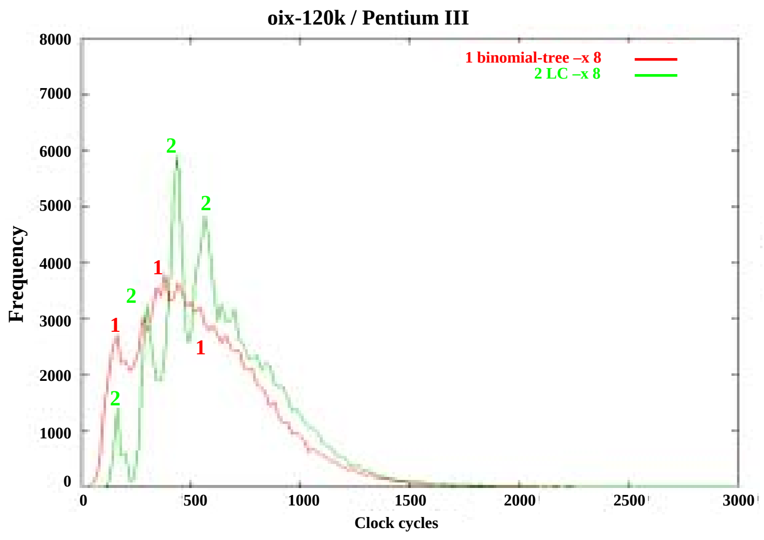

Figure 11: The proposed binomial scheme with one root and LC with an 8-level segmentation for oix routing table on Pentium III.

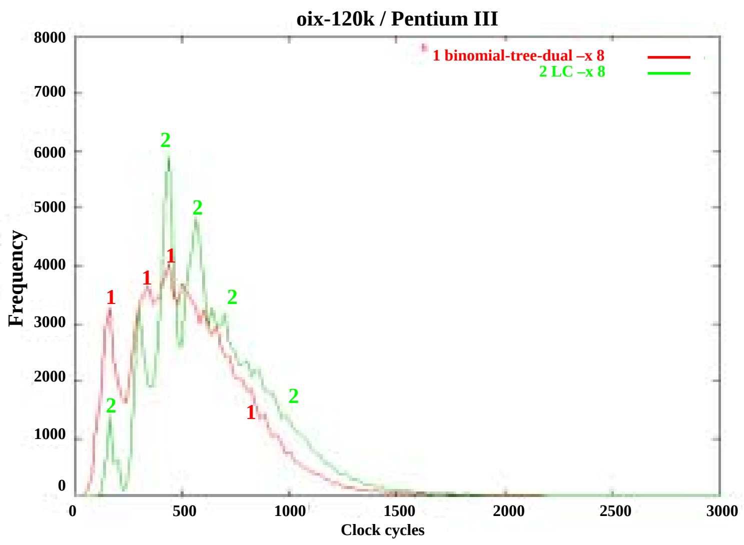

Figure 12: The proposed binomial scheme with dual roots and LC with an 8-level segmentation for oix routing table on Pentium III.

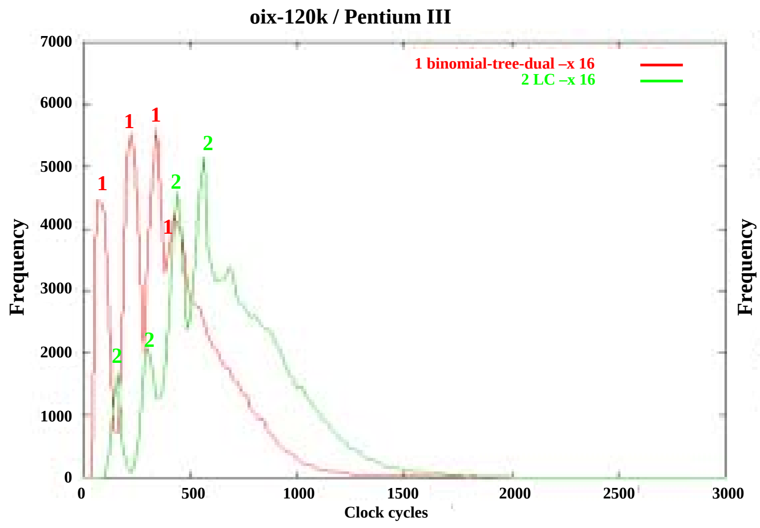

Figure 13: The proposed binomial scheme with dual roots and LC with a 16-level segmentation for oix routing table on Pentium III.

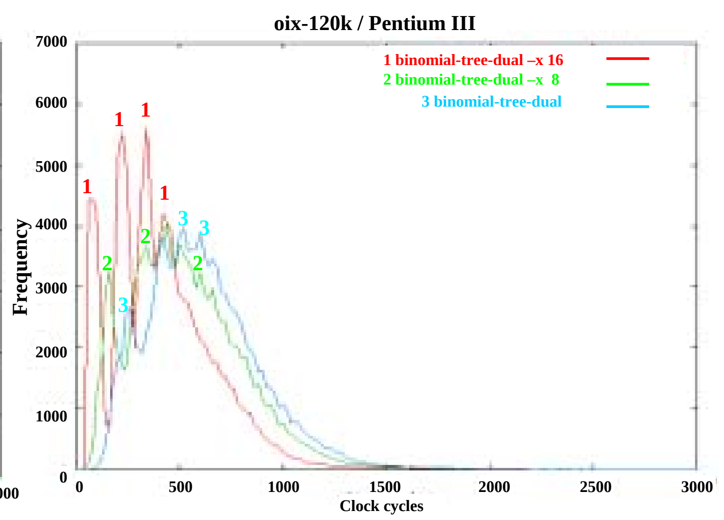

Figure14: The proposed binomial scheme with dual roots with 0-level, 8-level, and 16-level segmentations for oix routing table on Pentium III.

Scheme	Table1(120637 entries) clock	Table2(41709entries) clock
Binomial-x-0	1467	1165
Binomial-x-0-dual	1362	1064
Binomial-x-8	1287	875
Binomial-x-8-dual	1140	845
Binomial-x-16	827	528
Binomial-x-16-dual	682	483
Lc-trie-x-0	1208	587
Lc-trie-x-8	905	578
Lc-trie-x-16	879	557
Binary-trie-x-0	1989	1553
Binary-trie-x-8	1802	1357
Binary-trie-x-16	1395	1094

Table 1: Average cycle counts of the proposed binomial scheme with one root and dual roots, the binary trie, and the LC schemes with 0-level, 8-level, and 16-level segmentations for oix routing table on Pentium IV.