An Efficient IP Routing Lookup by Using Routing Interval

Pi-Chung Wang, Chia-Tai Chan, and Yaw-Chung Chen

Abstract: **Nowadays, the commonly used table lookup scheme for IP routing is based on the so-called classless interdomain routing (CIDR). With CIDR, routers must find out the best matching prefix (BMP) for IP packets forwarding, this complicates the IP lookup. Currently, this process is mainly performed in software and several schemes have been proposed for hardware implementation. Since the IP lookup performance is a major design issue for the new generation routers, in this article we investigate the properties of the routing table and present a new approach for IP lookups, our approach is not based on BMP and significantly reduces the complexity, thus the computation cost of existing schemes can be significantly reduced. We also propose an efficient IP lookup algorithm, with which we improve the binary search on prefixes to 30 millions of packets per second (MPPS) and 5,000 route updates/s under the same experiment setup with an even larger routing table.**

Index Terms: **Internet, gigabit networking, IP address lookup.**

I. INTRODUCTION

Speeding up the packet forwarding in the Internet backbone requires high-speed transmission links and high performance routers. The transmission technology keeps evolving and provision of gigabit fiber links is commonly available. Consequently, the key to increase the capacity of the Internet relies on in fast routers [1]. A multi-gigabit router must have enough internal bandwidth to switch packets between its interfaces and enough packet processing power to forward multiple MPPS [2]. Switching in the router has been studied extensively and solutions for fast packet processing are commercially available. As a result, the remaining major obstacle for the high performance router design is the slow, multi-memory-access IP lookup procedure. A router must search forwarding tables using the DA (destination address) as the key, and determine which table entry represents the best route to forward the packet to its destination. Since the development of CIDR in 1993 [3], IP routes have been identified by a ⟨routing prefix, prefix length⟩ pair, where the prefix length varies from 1 to 32 bits. Due to the fact that table entries have variable lengths and that multiple entries may represent the valid routes to the same destination, it may be time consuming to find out the best matching prefix (BMP), especially in a backbone router with a large number of table entries. It is extremely important that routing lookup scheme much be simple and fast for practical implementation in the next generation routers.

Manuscript received November 30, 2000; approved for publication by Norio Shiratori, Division III Editor, June 21, 2001.

P.-C. Wang and C.-T. Chan are with the Telecommunication Laboratories, Chunghwa Telecom Co., Ltd, Taipei, Taiwan, R.O.C, e-mail: pcwang@cht.com.tw, ctchan@cht.com.tw.

Y.-C. Chen is now with the Department of Computer Science and Information Engineering, National Chiao Tung University, Hsinchu, Taiwan, R.O.C., e-mail: ycchen@csie.nctu.edu.tw.

While designing the simple and fast routing lookup scheme, the following considerations should be considered simultaneously.

- **Simplicity:** The data structure and the routing lookup operation should be as simple as possible, such as using precomputation to avoid search complexity shown in [4].
- **Reducing routing-table space:** If we can reduce routing-table size to fit into the high speed SRAM, the performance can be promoted significantly.
- **Fast routing-table reconstruction:** The routing-table reconstruction is a major issue for the backbone router since it requires frequently route updates [5]. Basically, the update cost of the trie-based algorithm is the smallest. Other schemes, such as cache, hash and binary search, might potentially require full table refresh. To avoid this problem, one usually adopted the 16-bits pre-computation table. By dividing the whole address space into 64K small space, the problem can be alleviated.
- **Scalability:** The IP routing lookup algorithm must be scalable in the next generation network. Since the increasing growth of table size and the migration of IP addresses from 32 bits (IPv4) to 128 bits (IPv6) make the IP routing lookups be a challenging problem.
- **Parallelism:** To provide next generation tera-bit/s high-speed transmission, pipelining will play an important role. Therefore, another issue is to exploit the high parallelism within the routing-table lookups.

In this work, we introduce a new approach for IP table lookups, which can resolve the BMP problem. The proposed algorithm was initially inspired by the method in [4]. However, we show that the new algorithm is both more efficient and more flexible, and can be used to enhance the existing schemes significantly. According to the new approach, both the table construction time and table size of previous schemes can be significantly reduced. For example, we have improved previous scheme to achieve 30 MPPS and 5,000 route updates per second with the same experiment setup and an even larger routing table. With the state-of-the-art 700 MHz CPU, it can achieve more than 70 MPPS in average. The rest of the paper is organized as follows. In Section II, we describe the related works. Section III presents a new concept named as "Routing Interval" which can enhance the previous schemes significantly. We discuss our proposed IP lookup algorithm in Section IV. Section V outlines the experiment setup and results. Finally, a concluding remark is given in

Section VI.

II. RELATED WORKS

There has been a remarkable interest in the organization of routing tables during the last few years. The proposals include both hardware and software solutions. Degermark *et al.* [6] propose a trie-like data structure for quick routing lookups. It is able to compact a large routing table with 40,000 entries into a table with 150–160 kbytes size. The forwarding table is small enough to fit in on-chip cache memory. In hardware implementation, the minimum and maximum number of memory accesses for a lookup are two and nine, respectively. Gupta *et al.* presented fast routing-lookup schemes based on a huge DRAM [7]. The scheme accomplishes a maximum of two memory accesses for a lookup in a forwarding table of 33 Mbytes. By adding an intermediate-length table, the forwarding table can be reduced to 9 Mbytes; however, the maximum number of memory accesses for a lookup is increased to three. When implemented in a hardware pipeline, it can achieve one route lookup every memory access. This furnishes about 20 MPPS. Huang *et al.* [8] further improve it by fitting the forwarding tables into SRAM.

Regarding software solutions, algorithms based on tree, hash or binary search have been proposed. Srinivasan *et al.* [9] present a data structure based on binary tree with multiway branching. By using a standard trie representation with arrays of children pointers, insertions and deletions of prefixes are supported. However, to minimize the size of the tree, dynamic programming is needed. Karlsson *et al.* [10] solve the BMP problem by LC tries and linear search. Waldvogel *et al.* proposed a lookup scheme based on a binary search mechanism [11]. This scheme scales very well as the size of address and routing tables grows. The scheme requires a worst-case time of $\log_2$(address bits) hash lookups. Thus, five hash lookups are needed for IPv4, and seven for IPv6 (128-bit). This software-based binary search work is further improved by employing a cache structure as well as using multiway and multicolumn search techniques [4]. For a database of N prefixes with address length W, the native binary search scheme needs O(($W \times \log N$) searches. This improved scheme takes only O($W + \log N$) searches.

Although the existing works have their advantages, however, those approaches either use complicated data structures which result in high complexity for updating/building the forwarding table, such as [4], [9]–[11], or they are not scalable to fit in IPv6 [6]–[8].

III. ROUTING INTERVAL

As mentioned above, each IP address might be covered by multiple routing prefixes. However, only the longest one indicates the next-hop. From the geographical characteristic of the destination location, we investigate the routing table and propose a new routing concept based on the following observations. Before presenting our idea, it is necessary to define the segment for IP address. Each IP address is divided into two parts: segment (16 bits) and offset (16 bits).

The density of routing prefixes for segments is related to its hop-count: From the network topology, we know that all networks (including enterprise, campus, ISP) are interconnected through trunks. Since there are multiple trunks for neighbor cloud to connect with each other, the number of routing prefixes is large and with disorderly information. On the other hand, the routing prefixes are simple and much fewer for the remote cloud, i.e., farther geographical region.

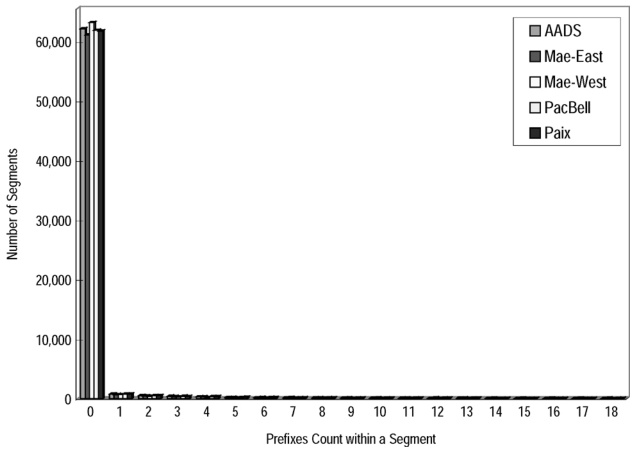


Fig. 1. Prefix count distribution.

Sparseness of routing prefixes: Although the number of Internet hosts increases exponentially, the number of routing prefix is still very sparse. For example, there are 65,536 segments but only about 56,000 routing prefixes in current backbone router (Mae-East 8/12/2000). There is less than one routing prefix per segment on average. We use the routing tables offered by the IPMA project [12], which provides a daily snap shot of the routing table used by some major Network Access Points (NAPs), as basis for our experiment. The number of prefixes whose length is longer than 16 bits in each segment is shown in Fig. 1. From the experimental results, we found that most of routing prefixes identify the next-hop only for few segments; we call this phenomenon as *routing locality*. It also reflects the property that most prefixes are related to neighbor clouds.

In a router, the number of possible next-hops for a segment is always much less than the total number of ports: The number of distinct next-hops in a routing table is limited by the number of other routers or hosts that can be reached in one hop, it is clear that these numbers would be small even in large backbone routers. For the enterprise or campus routers, the heavy use of default routes for outside destinations further reduces the routing table size and the number of next-hops. It leads that few routing prefixes are used to define the routes for most segments. Hence the number of next-hops for most segments is small and must be less than the number of related routing prefixes, as shown in Fig. 2.

Based on above discussion, we propose a new routing concept named as "Routing Interval" which fully utilizes the properties. By sorting the routing prefixes based on their lengths, we can build a new next-hop array in which each element maps to an IP address interval and is filled with related next-hop. For example, three routing prefixes 140.113.0.0/255.255.0.0/NH_1, 140.113.3.0/255.255.255.0/NH_2 and 140.113.215.0/255.255 .255.0/NH_3 with length 16, 24, 24, respectively. First, the

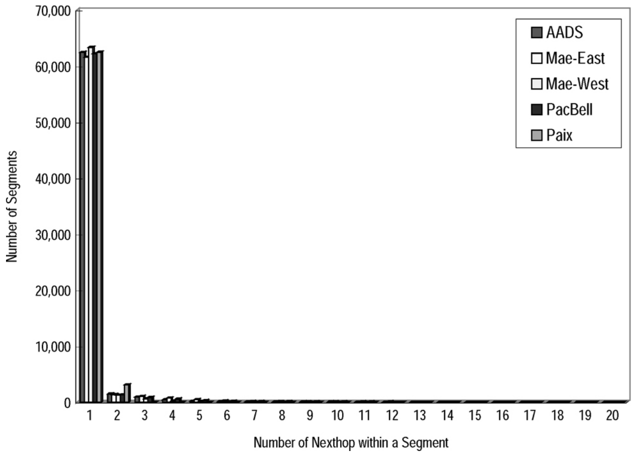


Fig. 2. The distribution of next-hop count within a segment.

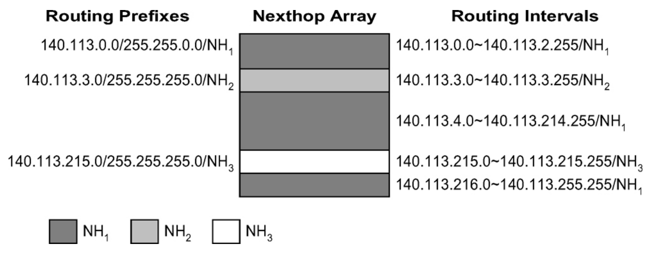

Routing Prefixes
Nexthop Array
Routing Intervals
140.113.0.0/255.255.0.0/NH1
140.113.3.0/255.255.255.0/NH2
140.113.215.0/255.255.255.0/NH3
140.113.0.0~140.113.2.255/NH1
140.113.3.0~140.113.3.255/NH2
140.113.4.0~140.113.214.255/NH1
140.113.215.0~140.113.215.255/NH3
140.113.216.0~140.113.255.255/NH1
NH1
NH2
NH3

Fig. 3. Routing interval example.

routing prefix 140.113.0.0/255.255.0.0/NH_1 creates an IP address interval 140.113.0.0 $\sim$ 140.113.255.255/NH_1. When the next routing prefix 140.113.3.0/255.255.255.0/NH_2 is processed, the interval is further partitioned into three regions, 140.113.0.0 $\sim$ 140.113.2.255/NH_1, 140.113.3.0 $\sim$ 140.113.3.255/NH_2, 140.113.4.0 $\sim$ 140.113.255.255/NH_1, respectively. According to this simple rule, we can use three routing prefixes to produce 5 elements based on the next-hop value and record each with ⟨begin address, end address, next-hop⟩. We named this tuple as "**Routing Interval**" as shown in Fig. 3.

Most of existing algorithms use different approaches to handle the BMP problem and result in either complex data structure or extra storage, or both. The most benefit to transform the routing prefix into routing interval is that the complexity of BMP problem can be avoided. For example, each node in Patricia tries consists of three pointers, one for prefix entry and two for child nodes due to possible longer match. Although various implementations of the Patricia tries have been proposed and the performance has been improved significantly, it still can be improved via applying the routing interval. The inefficient data structure can be simplified, with which each node may contain either a prefix pointer or two child pointers, but not both. With low transformation cost, we reduce the BMP problem into a much simpler search problem. The algorithm of transformation will be addressed below. Notice that our idea is completely different from the previous work [13], which concerns how to reduce the number of routing prefixes, while we focus on how to remove the BMP problem.

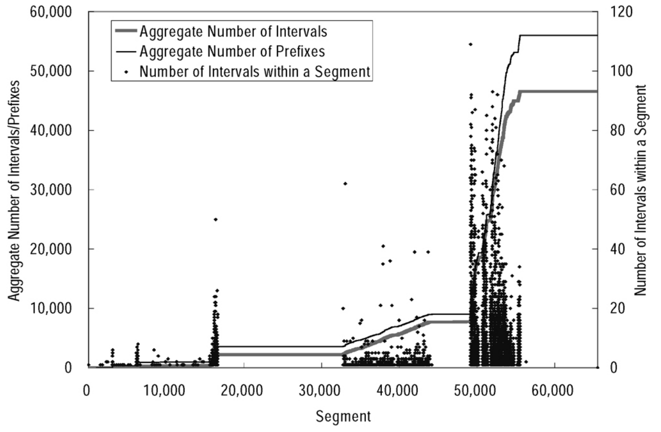


Fig. 4. Number of generated routing intervals for Mae-East.

Theoretically, the maximum number of generated routing intervals is $2N - 1$, where N is the number of routing prefixes (including default route). However, the experiments show different results. The result generated from the large routing table (Mae-East) is shown in Fig. 4. We found that the total number of routing intervals is less than that of routing prefixes in this case. Three reasons cause this: (I) For a region which indicates identical next-hop value, it has to be represented by several routing prefixes due to the limitation that the occupied region of routing prefixes must be the order of 2. (II) Some routing prefixes belong to the same routing interval, thus they can be merged into a single interval. For example, two routing prefixes, 140.113.0.0/255.255.0.0/NH_1, 140.113.3.0/255.255.255.0/NH_1, are with different prefix length and identical next-hop value. The routing prefix 140.113.3.0/255.255.255.0/NH_1 is redundant and can be removed. (III) There are zero-range intervals which can be eliminated. This phenomenon will be discussed in Section IV. The results of other NAPs are shown in Table 1, where the value following the slash is the number of routing prefixes. In the most numerical results, the generated numbers of intervals are fewer than that of the routing prefixes.

For most segments, the routing information, including the next-hops and routing prefixes, is very simple. Further, the distribution is related to the network topology. The routing information is more complex for the nearby region (in topology). We believe that the IP lookup scheme can be improved based on these characteristics, thus we convert the hierarchical routing prefixes into flat routing intervals. With the concept of routing interval, IP lookup complexity can be greatly reduced. We will address how existing schemes can take advantage of this concept below.

A. Qualitative Analysis and Enhancement with Existing Routing Schemes

Many existing algorithms use pre-computation to solve the BMP problem. We introduce three typical schemes and describe how they can be improved with routing intervals. Here, we only discuss how to reduce the complexity of algorithm in brief. In the next section, we will provide quantitative analysis in the as-

Table 1. Comparison of routing prefixes/routing intervals in different dates.

Date	AADS	Mae-East	Mae-West	PacBell	Paix
9/9/1999	9,287/35,769	44,233/47,784	28,853/28,998	22,263/26,278	9,341/9,270
12/22/1999	12,075/17,460	44,904/52,372	30,387/30,688	21,141/25,406	8,711/9,237
5/15/2000	24,451/23,515	42,800/53,226	30,492/33,514	23,619/29,959	8,944/11,587
8/12/2000	23,411/26,109	46,557/56,039	30,501/31,060	25,807/33,680	14,201/13,878

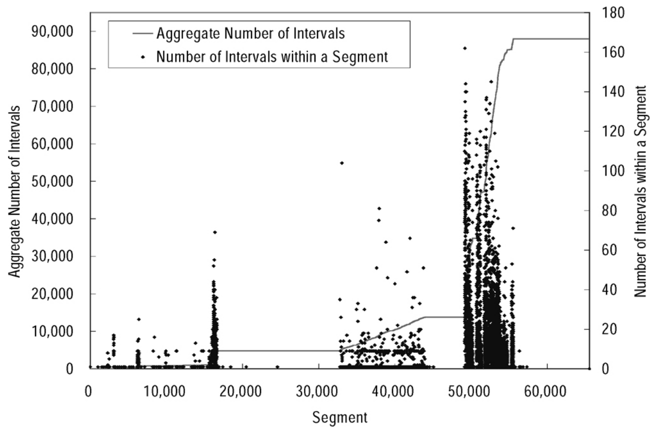


Fig. 5. Number of processed routing intervals for Mae-East.

pects of building time, lookup time and required storage through experiment.

The scheme proposed by Waldvogel *et al.* [11] performs the IP lookup based on the lengths of the prefixes. The prefixes with the same length are stored in a hash-table. The binary search is used to find out the longest prefix, and one hash-table lookup is performed at each step of this search. However, to find out the BMP with binary search is not trivial. For each prefix with length L, an extra marker must be added to the hash table with length 1 to $L-1$ to indicate that there exists a longer match, so that it will lookup the longer-prefix hash table. Furthermore, the best match prefix for the marker should also be calculated to avoid misleading. While we use the routing interval as the routing prefixes in this algorithm, it does not need to change the basic mechanism, but the pre-computation of best match for each marker can be removed. Since no misleading will happen with the usage of routing interval, there is no backtracking problem. Thus, we can save the pre-computation cost as well as simplify the data structure of forwarding table.

To migrate the routing prefixes to routing intervals here, we have to further divide the routing intervals so that the region of each interval can be represented with the power of 2. For example, an interval whose region is 6 will be cut into two intervals with region value 4 and 2, respectively. A sample result converted from Fig. 4 is shown in Fig. 5. The number of processed intervals would be a trade off between simplified constructing and searching efficiency.

Another scheme proposed by Srinivasan *et al.* [9] consists of two parts. In the first part, it reduces the number of distinct prefix lengths. With dynamic programming, the optimal expansion level can be calculated. Consequently, it constructs the tries based on the results. Leaf pushing and Cache line alignment are used to improve the performance. While applying routing intervals, the operation of leaf push can be eliminated because no longer match will occur. Each element in the node indicates a pointer to either the next node or the prefix table. In fact, leaf pushing is conceptually similar to our idea, but ours is more generalized. In [9], they also address how to improve the performance of binary search on hash table. By applying the routing interval concept, the enhancement discussed above can be applied, too.

In [10], Nilsson *et al.* use the skill called level compression (LC tries) to reduce the height of trie. The basic point is: since there is a fixed cost for comparing strings whose lengths are equal or less than the machine word, it is therefore more efficient to compare more bits at a time to reduce the number of comparisons and memory accesses. However, to improve the efficiency of data structure, they didn't handle the BMP problem in the trie. Each internal node keeps the number of nodes and the address of the leftmost one. The search only stops at leaf node, which contains the pointer to the prefix table. Once the pointed prefix doesn't match, it will jump to the shorter prefix of current one indicated in this entry. Therefore, it deals the BMP problem with linear search, this might result in poor performance in some cases.

Applying routing interval can eliminate the aforementioned problems. If we construct the trie with routing intervals, each interval will correspond to a leaf node in the trie. Once the search reaches a leaf node, the best matching will be found. Since no linear search is needed, the prefix table can be removed.

IV. PROPOSED IP LOOKUP SCHEME

We have introduced three enhancements to previous schemes and show the advantages of using the routing interval. Basically, it can simplify the data structure and reduce the pre-computation cost that leads to low table construction time. It might also reduce the required memory storage and the number of accesses in some cases. The transformation from routing prefix to routing interval is essential to design an effective routing scheme. Consequently, we propose an efficient transformation method, as well as an IP lookup scheme which further takes advantage of routing intervals. The proposed algorithm can handle 5,000 route updates per second, which includes the interval transformation and forwarding table construction.

A. Routing Interval Transformation

According to the routing interval concept, each routing prefix implies a routing interval that can be identified by just appending $(32 - prefix\ length)$ 0's and $(32 - prefix\ length)$ 1's as starting and ending address, respectively. For example, routing prefix $P_i = \langle 140.113.0.0/16/NH_i \rangle$ is with a routing interval $\langle S_i = 140.113.0.0, E_i = 140.113.255.255, NH_i \rangle$. However, it is an imprecise routing interval since it overlaps with other routing interval in the boundary; this causes the ambiguous routing lookup. Consequently, the major purpose of routing interval transformation is used to distinguish the routing intervals precisely.

The processing of each routing prefix will at most generate three routing intervals which are $\langle unknown, S_i - 1, unknown \rangle$, $\langle S_i, unknown, NH_i \rangle$ and $\langle E_i + 1, unknown, unknown \rangle$. The transformation algorithm is used to make up the unknown field. Basically, the transformation from routing prefix to interval is irreversible because the hierarchical information carried in routing prefix is removed. Each route update causes the re-transformation of routing interval. To ease the implementation, we adopt two link-lists, L_1 and L_2. The former is used to store the generated routing interval formats and initialized to be empty. The latter is used to store the possible routing interval $\langle E_i + 1, unknown, unknown \rangle$ and implemented as a stack, which performs "**push**," "**pop**," and "**modification**" operations.

With the sorted routing prefixes, let S_i, E_i and NH_i be the starting address, ending address and the next-hop of the routing interval P_i, respectively, where $S_1 \leq S_2 \leq \cdots \leq S_N$ (N is the number of routing prefixes, $1 \leq i \leq N$). At the beginning, we append the interval $\langle 0, unknown, default\ route \rangle$ to L_1 to L_1 and push the interval $\langle 255.255.255.255, 255.255.255.255,\ default\ \ route \rangle$ into L_2. To simplify the description, we use the subscript top and $rear$ to represent the starting address, ending address and next-hop of the top and rear routing interval in L_1 and L_2, respectively. For each routing prefix with imprecise routing interval$\langle S_i, E_i, NH_i \rangle$, we firstly check whether Stop (i.e., the starting address of the top interval in L_2) is smaller than $E_{rear} = S_{top} - 1$, then $E_{rear} = S_{top} - 1$. Then, we pop the top element, say, $\langle S_{top}, E_{top}, NH_{top} \rangle$, of L_2 and append it to the rear of L_1. After that, we update NH_{rear} with NH_{top}. We repeat these steps until there is no smaller interval in L_2, i.e., $S_{top} \leq S_i$, and modify E_{rear} to $S_i - 1$. It means that there exist an interval $\langle S_{rear}, S_i - 1, NH_{rear} \rangle$. Moreover, we must check whether S_{rear} is equal to S_i or not. If yes, then we replace NH_{rear} with NH_i. Otherwise, we append $\langle S_i, unknown, NH_i \rangle$ to L_1. Since it belongs to an unclear interval, we must further check whether S_{top} is equal to $E_i + 1$ or not. If $S_{top} = E_i + 1$, then we replace NH_{top} with NH_i. Otherwise, we push an interval, $\langle E_i + 1, unknown, NH_i \rangle$, into L_2. Finally, we pop all intervals stored in L_2 and append them to L_1 in sequence. The detailed algorithm is shown as below.

Routing Interval Transformation Algorithm

```
**Input:** $N$ routing prefixes.
**Output:** The set of ordered routing intervals.
Let $S_i$ and $E_i$ be the starting and ending address, $NH_i$ be the next-hop of $i_{th}$ routing prefix, respectively. (The subscript $top$ and $rear$ represent the starting address, ending address and the next-hop of the $top$ and $rear$ interval in $L_1$ and $L_2$, respectively.)
Let $P = P_1, P_2, \cdots, P_{N-1}$ be the set of sorted prefixes of an input segment.
For any pair of prefixes $P_i$ and $P_j$ in the set, $i < j$ if and only if $S_i < S_j$.
Append interval $\langle 0, unknown, default\ route \rangle$ into $L_1$ and push interval $\langle 1111 \cdots 1, 1111 \cdots 11, default\ route \rangle$ into $L_2$.
**For** $i = 1$ to $N$ **do**
  1. For the imprecise interval $\langle S_i, E_i, NH_i \rangle$ of $i_{th}$ routing prefix. Check if $S_i$ larger than $S_{top}$.
  2. If yes, modify $E_{rear}$ with $S_{top} - 1$. Pop the top interval from $L_2$ into $L_1$. Then modify $NH_{rear}$ with $NH_{top}$.
     Repeat step 1&2 until the result of comparison is false.
  3. Consequently, modify $E_{rear}$ as $(S_i - 1)$.
  4. Check if $S_{rear} = S_i$,
     4.a If yes, overwrite $NH_{rear}$ with $NH_i$.
     4.b Otherwise, append $\langle S_i, unknown, NH_i \rangle$ into $L_1$.
  5. Check if the $S_{top} = E_i + 1$,
     5.a If yes, overwrite $NH_{top}$ with $NH_i$.
     5.b Otherwise, push $\langle E_i + 1, unknown, NH_i \rangle$ into $L_2$.
Push all intervals stored in $L_2$ and append into $L_1$.
**For** all generated intervals **do**
  Merge two successive intervals with identical next-hop.
**Stop.**
```

In Fig. 6, we use an example to illustrate the process of the routing interval transformation. After processing the routing prefix P_3, the top element of L_2 and rear elements of L_1 are $\langle E_3, unknown, NH_3 \rangle$ and $\langle S_3, unknown, NH_3 \rangle$, respectively. For the routing prefix P_4, it firstly compares with S_{top}. Since S_{top} is smaller than S_4, E_{rear} is modified to S_i. It pops the top element, $\langle E_3 + 1, unknown, NH_3 \rangle$, of L_2 and append it to the rear of L_1. Then we modify NH_3 with NH_2. Since there is no smaller interval in L_2, we replace E_{rear} with $S_4 - 1$. It further checks whether S_{rear} is equal to S_4 or not. Because they are different, it must append the interval $\langle S_4, unknown, NH_4 \rangle$ to L_1. In addition, S_{top} is unequal to $E_4 + 1$, thus it pushes the interval $\langle E_4 + 1, unknown, NH_4 \rangle$ into L_2.

After processing all prefixes, the ordered intervals will be available in the array. We can perform another improvement by scanning all intervals again and merge two consecutive intervals with identical next-hop value. The time complexity of the transformation algorithm without prefix sorting is $O(M) = O(2N - 1) = O(N)$ where M is the interval count and N is the prefix count.

B. Modification of Binary Search for Speedup

In [4], Lampson *et al.* use two copies of the routing prefixes; one copy is padded with zeros and the other with ones, for IP lookup. With pre-computation, it can perform the prefix matching with binary search in a sorted array containing these extended prefixes. The required memory of forwarding table is about 1 Mbytes, which cannot fit into the L2 cache of most modern CPUs. Recall that a CPU READ to a byte will prefetch an entire cache line into the L2 cache, multiway search with cache line alignment are used to improve the performance. In spite of its scalability to IPv6, the route update will cause the reconstruction of forwarding table, which costs around 350 ms in the worst case.

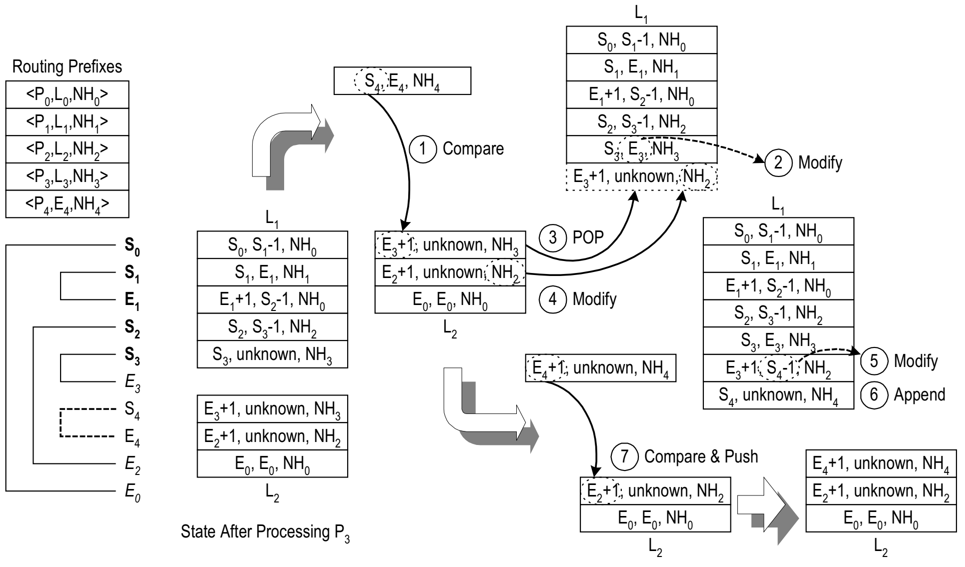


Fig. 6. An example of routing interval transformation.

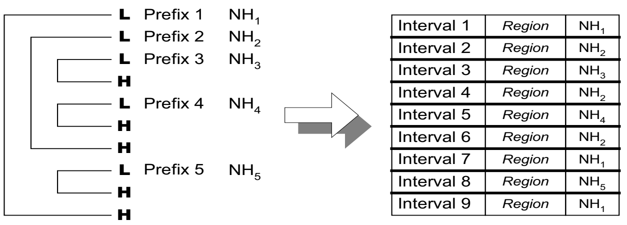


Fig. 7. Mapping from previous scheme into proposed scheme.

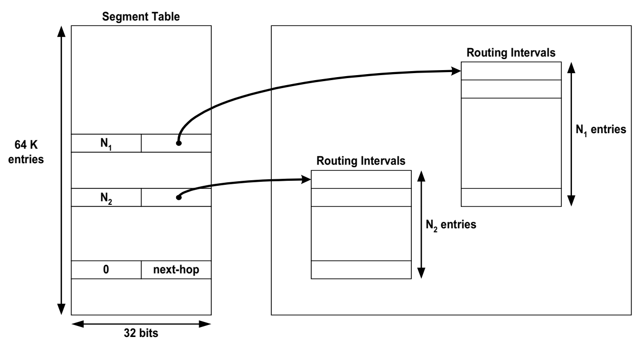


Fig. 8. Binary search with 16-bits segment table.

We claim that the performance can be greatly improved by adopting the routing interval. The mapping of previous scheme from routing prefix to routing interval is shown in Fig. 7. From the left part of Fig. 7, one can see that a native binary search can support IP lookup without any further pre-computation. Moreover, the number of entries (M) is always less than that of previous scheme ($2N$). However, the problem of route updates remains, there is no existing update technique that is faster than just building a table from scratch. Another problem resulted from the unbalanced interval distribution, as shown in Fig. 1, might degrade the effect of binary search.

To alleviate both problems, we split an IP address into two parts: the segment (16bits) and the offset (16bits), and add a segment table which extract the prefixes with length less than 16, as used in [4]. Each entry of the segment table consists of two fields: address of the first interval and the number of intervals, as shown in Fig. 8. If the interval count is zero, the value in another field indicates the next-hop. Otherwise, it will perform binary search to the intervals whose first 16-bits are the same. With 2 bytes for each field, the size of segment table will be 256 kbytes.

However, this scheme might not be helpful for future IPv6 routing table since we are unable to predict it. With address space, there might be 200,000 prefixes sharing the same first 16-bit stream while 100 prefixes sharing the other. Moreover, the 128-bit address is much longer than 32-bit machine word, i.e., 4 comparisons are required for each step. The multicolumn mechanism described in [4] can be applied to solve this problem. Since IPv6 is still some way off, we focus on how to enhance the lookup performance for IPv4.

C. Further Improvement with Memory Bus Alignment

To achieve higher performance, two requirements should be met. At first, the size of forwarding table should be small enough to fit into the L2 cache. By constructing the forwarding table, the total size including segment table is less than 480 kbytes for Mae-East routing table. Consequently, the data structure for binary search should comply with the memory bus. Unlike the cache line alignment described in the previous scheme, our improvement focus on the characteristic of memory bus between L1 and L2 cache. Most modern CPUs have larger bus width, such as 64bits, for access efficiency. Therefore, we merge two intervals (5bytes for each) into one entry with 8 bytes, as shown in Fig. 9. The binary search will test two routing intervals within one L2 cache access. The size of forwarding table can be reduced and the lookup speed can be made faster.

Table 2. Build, search speed, and memory usage complexity.

Algorithms	Build	Search	Memory
Patricia trie	O(NW)	O(W)	O(NW)
Binary search on hash tables	O($N\log_2 W$)	O($\log_2 W$)	O($N\log_2 W$)
Multibit trie	O(hW^2)	O(h)	O(hN)
LC tries	O(hN)	O(h)	O(hN)
Multiway search	O(N)	O($\log_M(2N)$)	O($2N$)
Proposed scheme	O(N)	O($\log_2(2N)$)	O($2N$)

N: the number of the prefixes, W: the length of the address,
h: the height of the trie, M: the number of branches

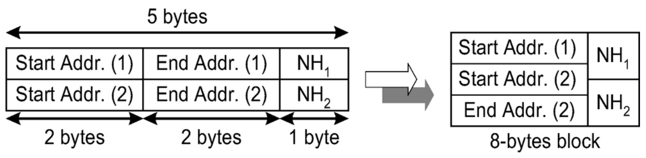


Fig. 9. Memory bus alignment for access efficiency.

D. Parallelism

We can further speedup the IP lookup by deploying parallel hardware. The basic idea is to distribute the IP lookups to the four memory banks equally. To achieve that, we have to record the referenced probability for each binary search table, and then sort the tables in terms of the referenced probability. Consequently, the tables are dispatched to each memory bank based on referenced probability sequentially. As shown in Fig. 10, we use four independent memory banks that can perform binary search separately. In the optimum situation, the performance can be improved almost four times. Also, with more memory banks, the IP lookup speed can be further increased.

E. Complexity

In Table 2, we show the complexity required for different software-based schemes. Note that these complexity measures do not indicate anything about the physical speed or actual memory usage. In the worst case, the number of routing intervals is twice of the number of routing prefixes. We will show the realistic performance metrics in the next section.

V. PERFORMANCE EVALUATION

To simplify the comparison with previous schemes, we need a comparable platform as used in previous work [4]. We choose a 300-MHz Pentium II running Windows NT that has a 512-kbytes L2 cache. We also use another 700MHz K7 CPU, to show what performance level we can achieve. Five routing tables from the IPMA project on August 12, 2000 are used for the experiment. We will show the performance of the proposed scheme with respect to three metrics: worst/average search time, storage and construct/update time.

To show the realistic performance of proposed algorithm, the simulation should avoid any L1 cache hit, for example, the result derived based on continuous address lookup will be improved externally since each required data might be located in L1 cache. To avoid this problem, we should adopt a larger search

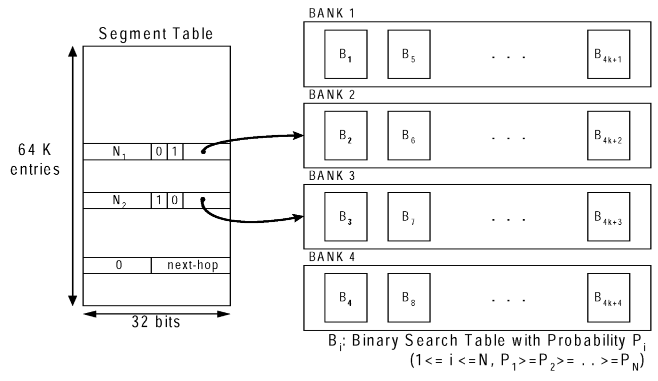


Fig. 10. Parallel IP lookup with four memory banks.

space. For example, to evaluate the worst-case performance, we choose the segment with maximum number of routing intervals (about 120). The required storage for this segment is 492 bytes after memory bus alignment. Then we make 256 copies of this segment with 123 kbytes totally, which is much larger than the size of L1 cache (32 kbytes). Consequently, we perform IP lookup for the address with maximum number of memory accesses around these 256 segments. As a result, we can make sure that each data is always fetched from the L2 cache. This method will be applied to our experiment for the practical result.

The detailed performance metrics are shown in Table 3. The proposed scheme works very well even with a large routing table. The update time represents the cost of table reconstruction for the segment with most of the intervals. Both build and update time are accounted as the cost of interval transformation. The proposed scheme can support at least 5,000 route updates per second which is much faster than 100 update/s as required in some BGP implementation [4]. The reason for the shortened build/update time is that our table construction is very simple. With link-list implementation, no large memory copy is needed in the interval transformation. Moreover, the construction of memory-bus-alignment array is much simpler than any existing schemes. The constructed forwarding table is small enough to fit into CPU L2 cache. Besides, the memory bus alignment helps to improve the access efficiency as well as decrease the number of entries with a factor of 2. For the routing table of Mae-East, the worst-case forwarding rate is larger than 4 MPPS. If we assign each IP address with the same weight, the average forwarding rate is about 30 MPPS. As used practical environment, we believe that the performance would be better due to the locality of

Table 3. Performance evaluation with five routing tables.

Performance Metrics	AADS	Mae-East	Mae-West	PacBell	Paix
Prefix count	26,109	56,039	31,060	33,680	13,878
Interval count	23,411	46,557	30,501	25,807	14,201
Memory required (kbytes)	374	459	393	372	327
Build time (msec)	93	141	109	110	62
Update time (μsec)	183.6	183.6	121.1	183.6	121.1
Average lookup time (nsec)	34	39	34	32	29
Worst case lookup time (nsec)	244	244	244	244	244

Table 4. Comparison with multiway search.

Performance metrics	Multiway search	Proposed scheme
Worst case lookup time (nsec)	330	244
Worst case update time (msec)	352	0.2
Build time (sec)	5.8	0.14
Memory required (kbytes)	950	459

Table 5. Comparison with other existing works.

Previous Schemes	Worst Case Lookup Time (ns)	Memory Required (kbytes)
Patricia trie	1,650	3,262
Binary search on hash tables	650	1,600
Lulea scheme	409	160
Multiway search	330	950
LC tries	$^{(*)}$500	464
Multibit trie	236	640
Proposed scheme	244	459

$^{(*)}$This is the average performance since the worst-case performance is not addressed.

Table 6. Performance evaluation with 700 MHz CPU.

Site	Memory Requried (kbytes)	Build Time (msec)	Update Time μsec	Average Lookup Time (nsec)	Worst Case Lookup Time (nsec)
Mae-East	459	70	117	22	153

traffic flows.

Consequently, we compare the proposed scheme with multiway search. The result generated from the routing table of Mae-East is shown in Table 4. Note that numbers measured on 200MHz Pentium Pro have been proportionately scaled to 300 MHz, as used in [4]. The critical build/update problem is significantly improved. This is mainly because the pre-computation cost for solving the BMP problem is removed in the proposed scheme. Moreover, the transformation from routing prefixes to intervals is performed cost effectively with link-list implementation. As a result, we can reduce the build/update cost significantly. Since no pre-computation information should be carried, our forwarding table is very simple and efficient. It can fit into the CPU L2 cache, thus results in better lookup performance. Therefore, even with slower L2 cache (150 MHz vs. 200 MHz), the worst-case lookup time is much shorter than that in multiway search. We didn't compare the average lookup time because it is not listed in [4].

In Table 5, we further compare other existing algorithms with ours. Again, we proportionately scaled all results of previous works to 300 MHz CPU to ease the comparison. The worst-case lookup time of LC tries [10] is not addressed in the literature so we fill it with average lookup time. Another important factor not shown in the table is the cost of route update that is not described in the papers on LC tries, binary search on hash tables, and Lulea compressed tries. However, all these schemes potentially require changing the complete data structure during a route update, they are likely to feature slow insertion/deletion times. In spite of the worst-case lookup time of multibit trie [9] is better than the proposed scheme, we used a larger routing table for experiment (56,039 vs. 38,816). Also, their worst-case update time (2.5 msec) is much larger than ours.

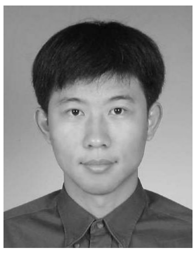
From the results of performance evaluation, the proposed scheme not only reduces the memory size significantly, outperforms the existing schemes in lookup speed, but also provides a fast routing-table update. Furthermore, it just uses simple data structure and routing lookup operations. Finally, we employ the AMD 700 MHz CPU to investigate the higher performance of the proposed scheme. As shown in Table 6, the average packet-forwarding rate is above 70 MPPS with 6 MPPS in the worst case. Thus, both build and update time benefit from the upgrade of CPU speed.

VI. FUTURE WORKS AND CONCLUSIONS

In this work we present a new concept named "routing interval" which can be used to resolve the BMP problem and significantly reduces the routing complexity. A routing interval transformation algorithm is proposed to support an efficient IP routing lookup. Via experiment, both the table construction time and table size can be reduced significantly. The proposed scheme not only reduces the memory size significantly, outperforms the existing schemes in lookup speed, but also provide a fast routing-table update. The constructed forwarding table is small enough to fit into the CPU L2 cache. By using the fast table-reconstruction algorithm, it is able to provide more than 5,000 route updates per second. This feature is preferred by the backbone router for handling the frequently route updates. With memory bus alignment, the scheme can further achieve 70 MPPS with 700 MHz CPU. These results demonstrate that the proposed scheme is efficient and flexible for IP packets forwarding. In the future work, we are trying to use intelligent data structure to further improve the performance of IP lookup and focus on hardware-based IP lookup scheme with high parallelism.

REFERENCES

[1] S. Keshav and R. Sharma, "Issues and trends in router design," *IEEE Commun. Mag.*, vol. 36, no. 5, pp. 144–151, May, 1998.
[2] C. Partridge *et al.*, "A 50-Gb/s IP router," *IEEE/ACM Trans. Networking*, vol. 6, no. 3, pp. 237–248, June, 1998.
[3] Y. Rekhter and T. Li, "An architecture for IP address allocation with CIDR," RFC, no. 1518, 1993.
[4] B. Lampson, V. Srinivasan, and G. Varghese, "IP lookups using multiway and multicolumn search," *IEEE/ACM Trans. Networking*, vol. 7, no. 4, pp. 323–334, June, 1999.
[5] S. Bradner, "Next generation routers. overview," in *Networld Interop*, 1997.
[6] M. Degermark *et al.*, "Small FORWARDING TABLES FOR FAST ROUTING LOOKUps," in *Proc. ACM SIGCOMM*, Sept. 1997, pp. 3–14.
[7] P. Gupta, S. Lin, and N. McKeown, "Routing lookups in hardware at memory access speeds," in *Proc. IEEE INFOCOM*, Mar. 1999, pp. 1240–1247.
[8] N. Huang, S. Zhao, and J. Pan, "A novel IP-routing lookup scheme and hardware architecture for multigigabit switching routers," *IEEE J. Select. Areas Commun.*, vol. 17, no. 6, pp. 1093–1104, June, 1999.
[9] V. Srinivasan and G. Varghese, "Fast IP lookups using controlled prefix expansion," *ACM Trans. Computers*, vol. 17, pp. 1–40, Feb. 1999.
[10] S. Nilsson and G. Karlsson, "IP-address lookup using $LC-Tries$," *IEEE J. Select. Areas Commun.*, vol. 17, no. 6, pp. 1083–1092, June, 1999.
[11] M. Waldvogel *et al.*, "Scalable high speed IP routing lookups," in *Proc. ACM SIGCOMM*, Sept. 1997, pp. 25–36.
[12] Merit Networks Inc., "Internet performance measurement and analysis (IPMA) statistics and daily reports," *IMPA Project*. Available at http://www.merit.edu/ipma/routing_table/.
[13] R. P. Draves *et al.*, "Constructing optimal IP routing tables," in *Proc. IEEE INFOCOM*, Mar. 1999, pp. 88–97.
[14] "High-speed routing table search algorithms," Torrent Networking Tech. Technical Paper. Available at http://www.torrentnet.com.
[15] S. Deering and R. Hinden, "Internet protocol version 6 (IPv6) specification," RFC., no. 1883, 1996.
[16] A. Tam, "How to survive as an ISP," *Networld Interop*, 1997.
[17] W. Doeringer, G. Karjoth, and M. Nassehi, "Routing on longest-matching prefixes," *IEEE/ACM Trans. Networking.*, vol. 4, no. 1, pp. 86–97, Feb. 1996.
[18] A. Moestedt and P. Sjodin, "IP address lookup in hardware for high-speed routing," in *Hot Interconnects VI*, Aug. 1998.



Pi-Chung Wang received his Ph.D. degree in computer science and information engineering from National Chiao Tung University, Hsinchu, Taiwan in 2001. He is now with the Telecommunication Laboratories Chunghwa Telecom Co,. Ltd. His research interests include the Internet multimedia communications, traffic control on high-speed network and L3/L4 switching technology. He is a member of IEEE.

Chia-Tai Chan received his Ph.D. degree in computer science and information engineering from National Chiao Tung University, Hsinchu, Taiwan in 1998. He is now with the Telecommunication Laboratories Chunghwa Telecom Co,. Ltd. His research interests include the design, analysis and traffic engineering of broadband multiservice networks.

Yaw-Chung Chen received his Ph.D. degree in computer science from Northwestern University, Evanston, Illinois in 1987. During 1987-1990, he worked at AT&T Bell Laboratories as a Member of Technical Staff. In August 1990, he joined the faculty of the Department of Computer Science and Information Engineering, College of Electrical Engineering and Computer Science, National Chiao Tung University, as an Associate Professor. He has been a Professor since 2000. His research interests include Internet traffic engineering, multimedia communications and high speed networking. He is a member of IEEE and ACM.