Update-aware Controlled Prefix Expansion for Fast IP Lookups

Yeim-Kuan Chang, Yung-Chieh Lin, and Kuan-Ying Ho
National Cheng Kung University, Tainan, Taiwan
{ykchang, p7894110, p7695139}@mail.ncku.edu.tw

Abtract-In high performance routers design, fast IP address lookup is always a challenge. In order to obtain fast lookup speed, multi-bit tries are often used to represent the routing tables [1,2,3,6]. The drawbacks of multi-bit tries are the large memory usage and extensive update cost. To reduce the memory usage of multi-bit tries, Srinivasan and Varghese proposed a scheme called Controlled Prefix Expansion (CPE) [2] that uses the dynamic programming technique to obtain the optimal multi-bit tries in terms of memory usage. Furthermore, current backbone routers usually run the Border Gateway Protocol (BGP). BGP may cause a few hundred of updates per second. To make multi-bit tries adequate to these updates, a series of multi-bit tries nodes need to be modified. Since these updates can seriously affect the lookup speed, we need to minimize these update cost. However, CPE does not concern this issue. In this paper, we explore the optimization issue in terms of the update cost. We want to find an update-optimal multi-bit tries that still have the efficiency of lookup speed and memory usage. Contrast to CPE, our solutions achieve a 30% reduction of the update overhead and improve 37% of the search speed. Besides, we also examine our schemes in IPv6 routing tables. The experimental results show that our scheme can also scale well in IPv6.

Keywords: IP address lookup, multi-bit tries, route update, controlled prefix expansion

1. Introduction

The advent of the World Wide Web (WWW) has doubled the network traffic on the Internet every few months. Besides, many emerging networking applications, such as video conferencing, remote distance learning, and digital libraries, are expected to create more and more traffic. If the Internet wants to continue to furnish good quality-of-service (QoS), four key issues must be addressed in the designing of the next generation IP routers: 1) higher link speeds; 2) better data throughput; 3) faster packet forwarding rate; and 4) quick adaptation to the routing changes. The solutions to the first two issues are now readily available. For example, fiber-optic cables can provide faster link speeds, and new IP-switching technology (layer-3 switching or multi-layer switching) can be used to transmit packet from the input interface of a router to the corresponding output interface at multi-gigabit speeds [6]. This paper deals with the other two issues: forwarding packets at high speeds while still allowing for frequent updates of the routing tables.

In order to achieve high packet forwarding rate, multi-bit tries are the well-know data structures that are often used to represent the routing tables [1,2,3,6]. Multi-bit tries can finish one lookup operation in a bounded number of memory references. However, the drawbacks of multi-bit tries are the large memory consumption and the poor update performance due to the routes changes. To reduce the memory consumption of multi-bit tries, many researchers have focused on the optimization issues of multi-bit tries in terms of the memory consumption [2,3]. In contrast, papers deal with the update-optimal multi-bit tries are rare. This paper focuses on finding the optimal multi-bit tries in terms of update costs for the fixed-stride multi-bit tries (FST) and the variable-stride multi-bit tries (VST), respectively. The rest of the paper is organized as follows. In section 2, we present an overview of previous works on IP address lookup. In section 3, we illustrate the concepts and properties of the proposed optimization techniques of multi-bit tries. The extensive experimental results are shown in section 4. Finally, a brief conclusion is remarked in section 5.

2. Related works

In backbone routers, IP address lookup is the most critical function in the packet forwarding process. Binary trie (i.e., 1-bit trie) [1,6] is the basic data structure that is used in the IP address lookup problem. Binary trie is efficient in the updates of routing routes. However, it has the worst lookup performance. To improve the lookup speed of the binary trie, a well-known data structure, multi-bit trie, is introduced [1,2,3,6]. Multi-bit trie can be divided into two categories, fixed-stride trie (FST) and variable-stride trie (VST). In FST, the sizes of the strides at the same level are the same. In contrast, the sizes of the strides at the same level can be different in VST. Building a multi-bit trie with a binary trie is actually choosing several level positions (i.e., bit positions) in a binary trie to perform expansion. How to decide these levels is the critical issue of constructing multi-bit tries. Basically, multi-bit tries with more expansion levels cost more memory accesses for the search or update operations, which means more search and update times; on the other hand, more expansion levels makes memory requirement of multi-bit tries smaller. The number of expansion levels simply becomes a trade-off between search speed and memory usage. Therefore, the current multi-bit tries optimization schemes [2,3] are only focusing on the instance: with a given number of expansion levels, finding the optimal multi-bit trie in terms of the memory consumption.

Srinivasan and Varghese [2] use controlled prefix expansion to construct the memory-efficient multi-bit tries. Given the maximum number of memory accesses allowed for one lookup operation (i.e., the maximum number of trie levels), they use dynamic programming to find the minimum total memory requirement for FST and VST, respectively. The dynamic programming techniques in [2] for FST work as follows. First, an auxiliary binary trie is constructed according to a given routing table. Let *nodes*(i) be the number of nodes at level i in the auxiliary binary trie. Let $T[j,r]$ be the optimal memory requirement for covering bit positions 0 through j by using r levels (assuming that the leftmost bit position is 0). Then $T[j,r]$ can be computed using dynamic programming. Thus, [2] generalized $T[j,r]$ to the following dynamic programming recurrence:

$$\begin{cases} T[j,r] = \min_{m\in\{r-2,\dots,j-1\}} \left[T[m,r-1] + nodes(m+1)\times 2^{j-m}\right] \\ T[j,1] = 2^{j+1} \end{cases}$$

The above recurrence terminates the (r-1)'th trie level at bit position m, such that it minimizes the total memory requirement. For prefixes at most W bits, we need to compute $T[W-1,k]$, where k is the number of levels in the trie being constructed. This algorithm takes $O(k\times W^2)$ time. For the case of IPv4, W=32.

Let r-VST be a VST that has at most r levels. Let $V[N, r]$ be the cost (i.e., memory requirement) of the best r-VST for a binary trie whose root is N. Sahni and Kim [3] proposed the following dynamic programming recurrence for $V[N, r]$:

$$\begin{cases} V[N,r] = \min_{s\in\{1\dots 1+height\,(N)\}} \left[2^s + \sum_{Q\in D_s(N)} V[Q,r-1]\right] \\ V[N,1] = 2^{height\,(N)+1} \end{cases}$$

where $D_S(N)$ is the set of all descendents of N at level s of N in binary trie and *height*(N) is the maximum level of which the trie rooted at N has a node. When more than one expansion level is permissible, the stride of the first expansion level may be any number s that is between 1 and *height*(N)+1. For any such selection of s, the next expansion level is level s of the binary trie whose root is N. The sum of equation gives the cost of the best way to cover all subtrees whose roots are at this next expansion level. Each such subtree is covered using at most r-1 expansion levels. It is easy to see that $V(R, k)$, where R is the root of the overall binary trie for the given prefix set P, is the cost of the best k-VST for P.

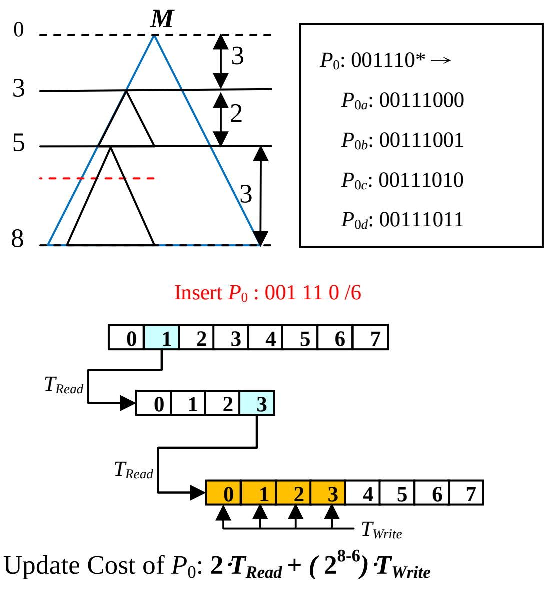

Figure 1: The update cost measurement of a 3-level multibit trie

3. Proposed Scheme

In this section, we first illustrate how we define the update cost for the insertion/deletion of a single prefix in a multi-bit trie. After that, we propose two recursive equations that apply the dynamic programming techniques for update-optimal FST and VST, respectively.

A. Measuring the Update Cost of inserting/deleting a Prefix

A multi-bit trie consists of a set of *strides*. A k-bit stride actually corresponds to a k-bit expansion in the binary trie, where k >= 1. A k-bit stride contains 2^k *elements*, where each element contains two fields, *next_ptr* and *next_hop*. The field, *next_ptr*, is a pointer that points to the stride of next level and the field, *next_hop*, is the next-hop information associated with the longest routing entry (prefix) that covers this element.

Since we want to obtain the update-optimal multi-tries, we need the metrics to define the update cost for the insertion/deletion of a single prefix. We first define two time units, $\boldsymbol{T_{Read}}$ and $\boldsymbol{T_{Write}}$. $\boldsymbol{T_{Read}}$ is the average time to retrieve the information of *next_ptr* or *next_hop* stored in one element. $\boldsymbol{T_{Write}}$ is the average time to modify the contents of *next_ptr* or *next_hop* stored in one element. Figure 1 illustrates how we measure the update cost when a single prefix is updated in a multi-bit trie. In Figure 1, the multi-bit trie M is a fixed 3-level trie for the prefixes whose length are at most eight. The first level of M only comprises one 3-bit stride (i.e., the root stride), the second level consists of 2-bit strides, and the third level consists of 3-bit strides. This also implies that the first level covers the first three bits of the prefixes (i.e., from bit position 0 to bit position 2, the leftmost bit position is bit position 0), the second level covers the following two bits (i.e., bit position 3 and bit position 4), and the last level covers the last three bits (i.e., from bit position 5 to bit position 7). Let P_0 = 001110* is a prefix that is going to be inserted into M. We assume during this insertion process there is no newly created stride. We first expand the length of P_0 from 6 to 8. As the result, P_0 is transformed into four length-8 prefixes, which are P_{0a} = 00111000, P_{0b} = 00111001, P_{0c} = 00111010, and P_{0d} = 00111011. We first use the first three bits of these four prefixes as the index (i.e., $(001)_2 = 1_{10}$) to obtain the *next_ptr* that is stored in the second element (whose index is 1) of the root stride. Then the insertion process goes to a second-level stride that is pointed by this *next_ptr*. At the second-level stride, bit position 3 and bit position 4 of the prefixes are used as the index (i.e., $(11)_2 = 3_{10}$) to obtain the new *next_ptr*. Following this new *next_ptr*, the insertion process arrives at a stride at the last level. After modifying the four *next_hop* fields whose indexes are 0, 1, 2, and 3 (correspond to the last three bits of P_{0a}, P_{0b}, P_{0c}, and P_{0d}), the insertion process for P_0 is finished.

Since the insertion process of P_0 retrieves two *next_ptr* from two different strides and modifies four *next_hop* of one stride, we define the update cost for this insertion is $2 \cdot \boldsymbol{T_{Read}} + (2^{8-6}) \cdot \boldsymbol{T_{Write}}$ (we assume index to one element only takes a slight time which can be ignored). On the other hand, the cost of deleting P_0 from M is exactly the same way as the case of insertion. Based on this measurement, we can calculate the update cost of any kind of multi-bit tries. The update-optimal multi-bit trie is the one with the minimal update cost.

B. Update Optimization For Fixed-Stride Tries

To insert/delete a prefix of length l in a fixed k-level multi-bit trie, the update process starts at the root stride and stops at a stride of level r such that the accumulative number of covered bits from the first level to level r is larger than or equal to l. Moreover, the accumulative number of covered bits from the first level to level r-1 is less than l. Let j be the accumulative number of covered bits from the first level to level r. As described before, the update cost for the prefix of length l will be $(r-1) \cdot T_{Read} + 2^{j-l} \cdot T_{Write}$. Furthermore, as *nodes*(i) defined in [2] represents the total number of nodes at level i in the auxiliary binary trie, we define $pfx(l)$ as the total number of prefixes whose prefix length are l in a given update trace. For a given r (# of levels of a multi-bit trie), let $Opt_u[j,r]$ represent a fixed update-optimal multi-bit trie that the accumulative number of covered bits from the first level to level r is j. We define the following recursive equations that can obtain the update-optimal r-level multi-bit trie.

$$\begin{cases} Opt_u[j,r] = \min\limits_{m \in \{r-1,\ldots,j-1\}} \left(Opt_u[m,r-1] + \sum\limits_{l \in \{m+1,\ldots,j\}} \left(pfx(l) \times \left((r-1) \times T_{\mathrm{Re}ad} + 2^{j-l} \times T_{Write} \right) \right) \right) \\ Opt_u[j,1] = \sum\limits_{l \in \{1,\ldots,j\}} pfx(l) \times (2^{j-l} \times T_{Write}) \end{cases}$$

The second equation is the terminative condition of the first recurrence. The number of level, r, is set to 1 in the second equation means that the second equation only calculates the update cost for a 1-level multi-bit trie (i.e., a array consists of 2^j elements). Hence, the update cost for the prefix with length l will be $2^{j-l} \cdot T_{write}$. Thus, the total update cost for this 1-level multi-bit trie can be obtained by simply summing up the update costs of each kind of prefix length. A variable, m, divides the first recursive equation into two parts, $Opt_u[m,r-1]$ and $\sum\limits_{l \in \{m+1,\ldots,.j\}} \left(pfx(l) \times \left((r-1) \times T_{\mathrm{Re}ad} + 2^{j-l} \times T_{Write} \right) \right)$, where m is the accumulative number of covered bits from the first level to level (r-1). This means the update cost of $Opt_u[j,r]$ has been divided into the update cost of $Opt_u[m,r-1]$ plus the update cost of the prefixes whose length are larger than m. The update costs for a prefix whose length is larger than m will cost (r-1) times of T_{Read} and 2^{j-l} times of T_{Write}. Hence, the total update cost for the prefixes whose length are larger than m is $\sum\limits_{l \in \{m+1,\ldots,.j\}} \left(pfx(l) \times \left((r-1) \times T_{\mathrm{Re}ad} + 2^{j-l} \times T_{Write} \right) \right)$. Then the recurrence goes to solve the problem of $Opt_u[m,r-1]$ and repeats the recurrence again and again till it encounters the terminative condition. Figure 2 shows the pseudo code that applies the dynamic programming techniques to solve the proposed recursive equations. For any given IPv4 update trace, since the maximum possible prefix length of IPv4 is 32, variable j is initially set to 32. For a given r (# of levels of the multi-bit trie), by the initial invocation of FST_Opt_UpdCost(32,r) (i.e., $Opt_u[32, r]$), Figure 2 will return a r-level multi-bit trie that has the minimal update cost.

```
Algorithm FST_Opt_UpdCost (j, r)
// Initial invocation is FST_Opt_UpdCost (W, r),
// W is maximum possible length of prefixes
// r is a given number that represents the number of level
// Return a r-level FST with the minimal update cost
{   min = ∞
    if summation[ pfx(1) to pfx(j) ] = 0
       then return 0
    else {
       if r = 1
        then return Summation{pfx(i)×(2^{j–l}×T_Read ), for i←1 to j}
       else {
          then for(m: r-1 to j-1) {
             cost = FST_Opt_UpdCost(m, r–1)
             + Summation{ pfx(i), for i←m+1 to j } × (r–1) × T_Read,
             + Summation{ pfx(i) × 2^{j–l} × T_Write },  for i←m+1 to j
             if min > cost
                min = cost
             else
                then continue
          }
          then return min
       }
    }
}
```

Figure 2: Algorithm FST_Opt_UpdCost

C. Update Optimization for Variable-Stride Tries

We now consider how to obtain the update-optimal variable-stride multi-bit trie (VST). For simplicity, assume that we have built an auxiliary binary trie, and we wish to convert this binary trie into an upate-optimal VST.

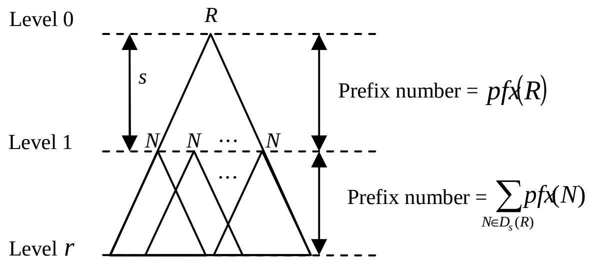

Figure 3: The stride s partitions a binary trie T whose root is R into one expansion level and several subtrees rooted at N, where $N \in D_s(r)$

Database	Number of prefix	Number of 24-bit prefixes	Percentage of Prefix in 24-bit
canada	157118	85938	54.6%
as120k	127071	69678	54.8%
oix-2002-4	124824	68978	55.2%
oix-2005-4	163574	85305	52.1%
funnet	41709	25206	60.4%

Table 1: The stats of the IPv4 BGP routing tables

Unlike FST, we first choose the size of strides at the last level of FST. In VST, we first choose the size of stride at the first level of VST (i.e., the size of the root stride of VST). Let *R* be the root of the auxiliary binary trie *T*. We start with the binary trie *T* (Figure 3) and select an *s*-bit stride as the root stride to partition the binary trie into 2^s subtries. Let $D_l(R)$ be the level *l* (the root (*R*) is at level 0) descendents of the root of the root *R* of *T*. Note that $D_0(R)$ is just R and $D_1(R)$ is the children of *R*. When the trie is partitioned with stride *s*, each subtrie *ST*(*N*), rooted at node $N \in D_S(R)$ defines a partition of the routing table. Note that $0 < s \leqq T.height + 1$, where *T.height* is the height of *T* (i.e., maximum level at which there is a descendent of *R*). When $s = T.height + 1$, $D_S(R) = \phi$. Let $pfx_l(N)$ be number of prefixes at level *l* of the *ST*(*N*), and *pfx*(*N*) denote number of prefixes those reside at *ST*(*N*) (i.e., those prefixes covered by *N*).

To use recursive partition effectively, we must select the stride s appropriately for each expansion level. For this selection, we set up a dynamic programming recurrence. As that scheme we used for solving update-optimal FSTs, given the number of expansion level r, this recurrence calculates expectation update cost of all *r*-VST to obtain the best VST with smallest cost. To calculate update cost of a *r*-VST, which actually the amount of update prefixes expanded to, we use prefix numbers of a level instead of prefix probability of a length. Because in a VST every trie-node has different strides, makes prefixes at same level in binary tire could be expanded to different number of element in VST. We need to the information of prefix numbers that covered by each subtrie in the binary trie *T*.

Let *Opt*(*N*, *j*, *r*) be the minimum update cost by a recursively partitioned representation defined by levels 0 through j of *ST(N)* (i.e., the subtrie of *T* rooted at *N*). At beginning, *j* is *height*(*N*) merely. From the definition of recursive partitioning, the choices for stride s in *Opt*(*N*, *j*, *r*) are 1 through *j*-1. The update cost of prefixes at first *s* levels at *ST*(*N*) is *Opt*(*N*, *s*, *1*), and that of the rest prefixes is prefix number multiple one memory read time(i.e, T_R) plus update cost of each subtrie. When *r* = 1, *ST*(*N*) is directly converted to an expanded array, so minimum update cost, i.e., *Opt*(*N*, *j*, 1) is totally the amount of prefixes expansion. Therefore, from the definition of recursive partitioning, we obtain following dynamic programming recurrence for *Opt*(*N*, *j*, *r*):

$$\begin{cases} Opt(N,j,r) = \min\limits_{s\in\{1,\ldots,j-r+1\}} \left\{ Opt(N,s,1) + \sum\limits_{Q\in D_S(N)} \left(Opt(Q,j-s,r-1) + pfx(Q)\cdot T_{Read} \right) \right\} \\ Opt(N,j,1) = \sum\limits_{l\in\{1\ldots j\}} pfx_l(N) \times \left(2^{j-l}\cdot T_{Write} \right) \end{cases}$$

	CPE	UCPE(3,1)	UCPE(1,1)
4-level	16-21-24-32	17-21-24-32	17-21-24-32
5-level	16-20-22-24-32	16-20-22-24-32	17-21-24-26-32
6-level	12-17-20-22-24-32	16-20-22-24-26-32	17-21-24-26-27-32
7-level	11-16-18-20-22-24-32	16-20-22-24-26-27-32	17-21-24-26-27-28-32
8-level	8-12-16-18-20-22-24-32	16-20-22-24-26-27-28-32	17-21-24-26-27-28-29-32

Table 2: The chosen partition levels of various schemes for IPv4 routing tables

Database	Number of prefix
AS64471	888
AS2.0	893
Generated1 (5000_table)	5108
Generated2 (10000_table)	10088
Generated3 (big_table)	20070

Table 3: The stats of five IPv6 routing tables

4. Experiment Results

A. Environment

We programmed our dynamic programming algorithms and the original CPE algorithm [2] in c code, and all experiments were conducted on a 2.4-GHz Pentium IV PC with 512KB L2 Cache and 1 GB DDR RAM. All programs were compiled by using the GCC-3.3.2 compiler and optimization -level 03 is used.

B. Results for IPv4

Our experiments were conducted by using five real IPv4 BGP routing tables (obtained from [4] and [5]). Table 1 shows the stats of these five routing tables. To calculate the update cost, we need the update traces for these five tables. We first analyze the update traces obtained from [8] then generate the appropriate update traces for each tables. These synthetic update traces are proportional to those we analyzed from [8]. According to the synthetic update traces, we can obtain the stats of *pfx*(*l*), where $1 \leq l \leq 32$. Since the trends of the experimental results for five IPv4 tables are almost the same, here we only show the results for the table named *canada*.

Table 2 shows the update-optimal FST that are generated by the proposed recurrences in different levels. The notations, UCPE(3,1) and UCPE(1,1), represent the proposed update-optimal CPE that are measured by using the ratio of T_{Write} and T_{Read} in 3 : 1 and 1 : 1, respectively. As we can see in Table 2, the result of our UCPE(3,1) in 7 levels is 16-20-22-24-26-27-32. These seven numbers represents the seven different levels of the auxiliary binary trie (the root node of the binary trie is at level 0). It implies that the root stride is a 16-bit stride and the strides at the second level of this 7-level FST are 4-bit (20-16=4) strides (expanded by the 17’th bit to the 20’th bit). The sizes of the strides at other levels can be derived in the

	CPE	UCPE(3,1)	UCPE(1,1)
4-level	24-36-48-64	28-38-48-64	28-38-48-64
5-level	20-29-38-48-64	24-34-40-48-64	24-34-40-48-64-
6-level	18-25-32-40-48-64	24-32-38-42-48-64	24-32-38-42-48-64
7-level	17-23-29-34-40-48-64	24-32-36-40-42-48-64	24-32-36-40-42-48-64
8-level	16-22-27-32-37-42-50-64	24-32-36-38-40-42-48-64	24-32-36-38-40-42-48-64
10-level	9-19-23-27-31-31-38-42-50-64	16-28-32-36-38-40-42-44-48-64	24-32-36-38-40-42-46-48-49-64

Table 4: The chosen partition levels of various schemes for IPv6 routing tables

same way. In Table 1, it shows that prefixes with length 24 are the majority in the routing tables. Moreover, in [8], it reveals that the update frequency of prefixes with length 24 is the highest among all prefix length. Hence, to minimize the update costs due to the length-24 prefixes, our UCPE will certainly expand the strides to level 24 of

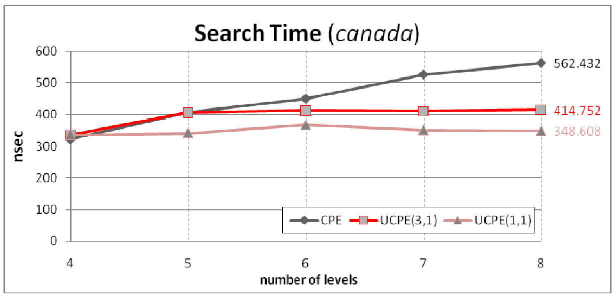

Figure 4(a)

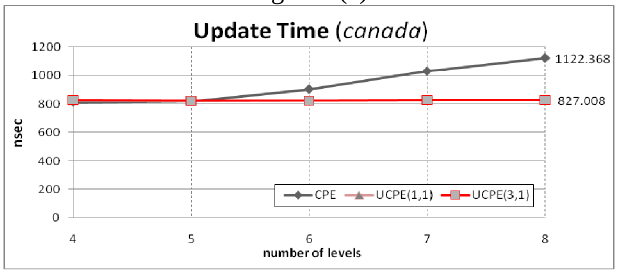

Figure 4(b)

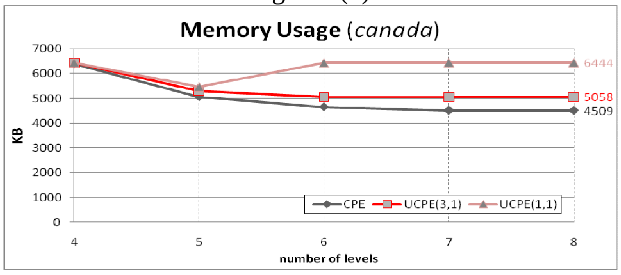

Figure 4(c)

Figure 4: Performance of FSTs designated by CPE and proposed UCPE in terms of (a) time requirement for search (in nsec), (b) time requirement for update (in nsec), and (c) memory requirement for update (in KBytes)

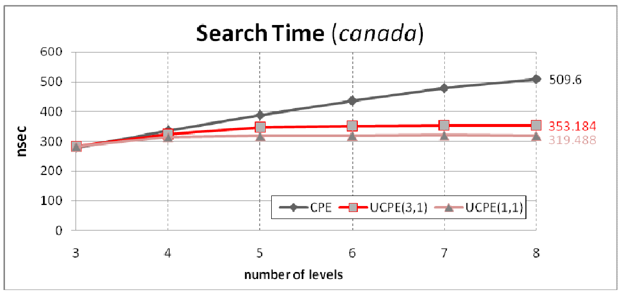

Figure 5(a)

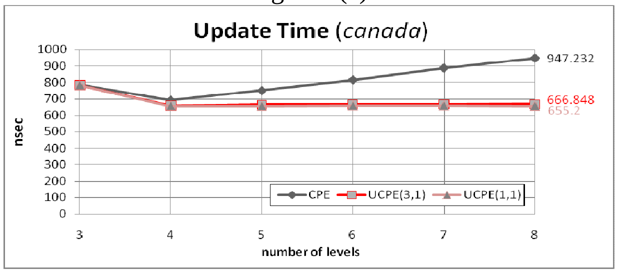

Figure 5(b)

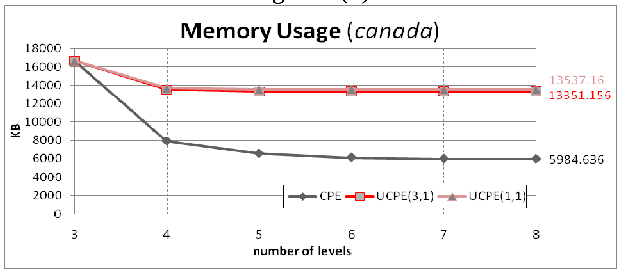

Figure 5(c)

Figure 5: Performance of VSTs designated by CPE and proposed UCPE in terms of (a) time requirement for search (in nsce), (b) time requirement for update (in nsce), and (c) memory requirement for update (in KBytes)

the auxiliary binary tries. In contrast, in order to minimize the memory cost due to the expansion of length-24 prefixes, CPE [2] will also certainly expand the strides to level 24 of the auxiliary binary trie. This is the reason why CPE [2] and the proposed UCPE will all choose level 24 of the binary trie to expand the strides in Table 2.

Figure 4 and Figure 5 show the experimental results in terms of search time, update time, and memory usage for FST and VST, respectively. As we can see, with the increase of the number of level of multi-bit tries, the search speed of our UCPE is more efficient than CPE. This is due to our UCPE expands more bits at each level of multi-bit trie than CPE [2]. Take the sixth row of Table 2 for example, assume the length of the longest matching prefix for an incoming packet is 24. It needs 7 memory accesses for this packet to find the longest matching prefix at the 7'th level of 8-level CPE. However, it only needs 4 memory accesses in our 8-level UCPE(3,1) and 3 memory accesses in our 8-level UCPE(1,1). This is the reason why our UCPE has faster search speed than CPE [2]. Since our UCPE is the update-optimal multi-bit tries, our UCPE uses 26% less update time than CPE when the level of multi-bit tries is 8. However, since CPE is the

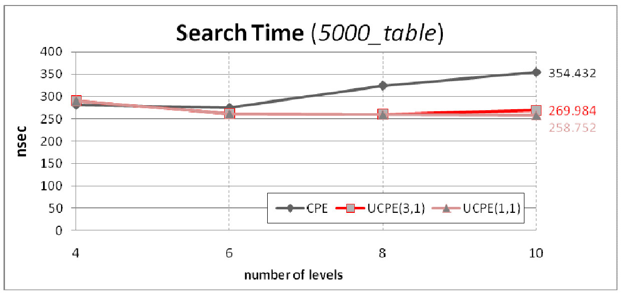

Figure 6(a)

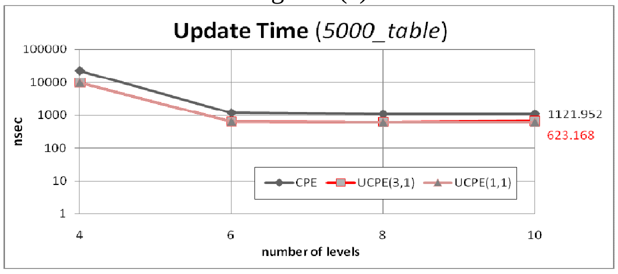

Figure 6(b)

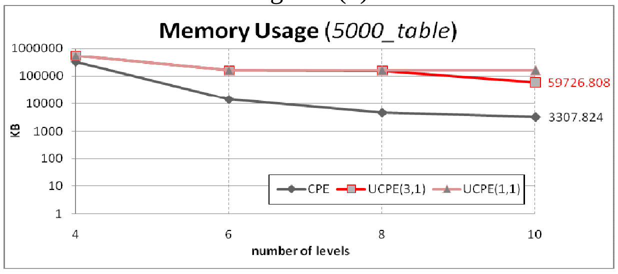

Figure 6(c)

Figure 6: Performance of IPv6 FSTs designated by CPE and proposed UCPE in terms of (a) time requirement for search (in nsce), (b) time requirement for update (in nsce), and (c) memory requirement for update (in KBytes)

memory-optimal multi-bit tries, our UCPE(3,1) and UCPE(1,1) require extra 1.4 and 1.9 Mbytes memory usage than CPE in the worst case, respectively.

C. Results for IPv6

Table 3 shows the stats of five IPv6 routing tables we used. The first two tables (AS6447 and AS2.0) are the real IPv6 routing tables that are obtained from [5]. As we can see, the current IPv6 tables are still very small, tables contain more than 2000 entries are rare. Thus, we use the methodology proposed in [7] to generate three large IPv6 tables (Generated_1, Generated_2, and Generated_3). As in IPv4, we analyze the update traces obtained from [8] and generate the synthetic update trace for each IPv6 table. Although the address space of IPv6 is 128 bits, there are almost no tables contain prefixes with length 128. That is due to the last 64 bits of the IPv6 addresses are the MAC addresses which are not able to be obtained. As the results, we only deal with the first 64 bits of any prefixes. Table 4 shows the experimental results for r-level FST, where $4 \leq r \leq 10$. As we can see in Table 4, our UCPE expand more bits than CPE [2] at each level of r-level FST. Hence, our UCPE needs less memory accesses to find the longest matching prefix for every incoming packet. As the results, our UCPE has better search time than CPE [2]. Moreover, since prefixes with length 32, 38, and 40 are the prefixes that have the high update frequency, our UCPE usually naturally expand the strides to the 32'th, 38'th and 40'th levels of the auxiliary binary trie. Since the trend of IPv6 experimental results are like the results in IPv4, here we only use Figure 6 to show the performance comparison of r-level FST for IPv6 table, Generated 1 (5000_table). In a 10-level FST, our UCPE uses 27% and 44% less time than CPE [2] for search and update, respectively. However, the drawback of our UCPE is that the memory usage is proportional to the longer address space of IPv6.

5. CONCLUSION

Our UCPE optimization schemes reduce the update overhead of multi-bit tries without losing the fast lookup speed. Compared with the previous scheme Controlled Prefix Expansion (CPE) [2], the proposed solutions provide a faster lookup and update speeds. In IPv4 lookup, we have reduced update cost reach to 30 % when the worst case memory access is bounded to 8. Beside, our solutions also provide a faster lookup speed than CPE in general case. For multi-bit tries with large number of level, the improvement of update and search speed by our UCPE are more significant. We also tested our UCPE scheme with IPv6 routing tables. As the results of IPv4, the performances of update and search are improved by our UCPE. However, the memory usage of our UCPE appears to be much larger than CPE. This is expectable because the address space of IPv6 is much longer than IPv4. This case makes the proposed UCPE able to be improved by considering the constraint of the memory usage constraint. We considered this issue as a future works of this paper.

References

[1] M.A.Ruiz-Sanchez, E.W.Biersack, and W.Dabbous, "Survey and Taxonomy of IP Address Lookup Algorithm," *IEEE Network*, Vol.15, pp.8- 23, 2001.

[2] V. Srinivasan and G. Varghese, "Fast address lookups using controlled prefix expansion," *ACM Transactions on Computer Systems*, vol. 17, no. 1, pp. 1–40,Feb. 1999.

[3] S. Sahni and K. S. Kim, "Efficient Construction Of Multibit Tries For IP Lookup," *IEEE/ACM Transactions on Networking*, vol.11, no.4, pp.650-662, 2003.

[4] D. Meyer, University of Oregon Route Views Archive Project, at http://archive.routeviews.org/.

[5] BGP Routing Table Analysis Reports, http://bgp.potaroo.net/.

[6] H. Chao, "Next Generation Routers," *Proceeding of the IEEE*, vol. 90, no. 9, pp. 1518-1558, September 2002.

[7] K. Zheng and B. Liu, "V6Gene: a scalable IPv6 prefix generator for route lookup algorithm benchmark," *proceeding of AINA* 2006, pp. 147-152, April 2006.

[8] RIPE Network Coordination Centre, http://www.ripe.net/.