Power-Efficient TCAM Partitioning for IP Lookups with Incremental Updates

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. Ternary Content-Addressable Memories (TCAMs) provide a fast mechanism for IP lookups and a simple management for route updates. The high power consumption problem can be resolved by providing a TCAM partitioning technique that selectively addresses smaller portions of a TCAM. This paper proposes a 2-level TCAM architecture using prefix comparison rule to partition the routing table into a number of buckets of possibly equal sizes. The 2-level architecture results in a 2-step lookup process. The first step determines which bucket is used to search the input IP. The second step only searches the IP in the determined bucket from the first step. The prefix partitioning algorithm provides an incremental update. Experiments show that the proposed algorithm has lower power consumption than the existing TCAM partitioning algorithms.

1 Introduction

Backbone routers have to forward millions of packets per second at each port. The IP lookups of the routers becomes the most critical operation to reach the capability of forwarding millions of packets per second. In [1], a large variety of routing lookup algorithms were classified and their worst-case complexities of lookup latency, update time, and storage usage were compared. However, these schemes using DRAMs or SRAMs can hardly meet the wire speed requirements needed for current terabit router design since they usually need many memory access cycles to look for a matched routing entry.

TCAMs are fully associative memories that allow a ”don’t-care” state for each memory cell, in addition to the states of 0 and 1. Each entry of a TCAM consisting of multiple cells is long enough to store a prefix for IP lookups or a rule for packet classification. TCAM is designed in such a way that all the entries are looked up in parallel against the incoming IP address. Thus, a matched entry if it exists can be found in a single TCAM access cycle. If multiple entries match the IP address, the entry with lowest address (i.e., longest prefix) in TCAM is typically returned as the result. Moreover, the update process in TCAMs is in general very simple [5].

There are two major disadvantages for TCAMs, the high cost-to-density ratio and power consumption. TCAM designs from IDT and Netlogic have effectively solved the issue of high cost-to-density ratio. The cost of their TCAM designs is very competitive with other hardware alternatives.

The high power consumption comes from the fact that the hardware circuits of all the entries in a TCAM are activated in parallel to perform the matching operations. Therefore, TCAM vendors today provide entry selection mechanisms to reduce power consumption by selecting fixed regions of TCAM called buckets for matching process.

In this paper, we propose a prefix partitioning scheme that performs better than subtree-split and postorder-split in most of the cases. The proposed scheme is based on the prefix comparison mechanism by which we can compare two prefixes of different lengths. Using the prefix comparison mechanism, we can sort the original prefixes and divide them evenly into K groups by selecting $K-1$ pivot prefixes with some small number of duplicated pivot prefixes. We will show by performance evaluation on real routing tables that the proposed partitioning scheme performs better than subtree-split and postorder-split in power consumption reduction.

The rest of the paper begins with the definition of IP lookup problem and related works in section 2. Section 3 illustrates the basic ideas of the proposed partitioning algorithms and the detailed design. The results of performance comparisons using real routing tables available on the Internet are presented in section 4. Finally, a concluding remark is given in the last section.

2 Problem Definition and Related Works

A TCAM consists of a large number of multi-cell entries. Each cell of an entry in a TCAM is a ternary bit which is implemented by a value bit and a netmask bit. A cell is compared with the corresponding bit in the target IP. A cell match is found if the netmask bit is 0 or the value bit is the same as the corresponding bit in the target IP. A TCAM entry matches the target IP if all the cells match the target IP. The TCAM is designed in such a way that when an IP is input, all the TCAM entries are activated to compare against the input IP. Only the match in the TCAM entry with the lowest address is returned as the final result. To ensure the TCAM entry with the lowest address is the longest prefix match, the prefixes in the TCAM must be stored in order of decreasing prefix length. The prefix length ordering slows down the TCAM update process. Shah and Gupta [5] have proposed efficient update schemes for TCAM updates.

In [4], Zane, et. al, took advantage of the entry selection mechanisms and developed two sets of prefix partitioning schemes to reduce the number of prefixes searched in an IP lookup. Their first partitioning scheme called bit selection architecture combines simple glue logic with TCAM. The glue logic uses a simple hashing function to select a set of input bits called hashing bits as an index to the appropriate TCAM bucket. The bit selection schemes are not suitable for real implementations because of the following drawbacks. The first drawback is that it only deals with prefixes of length 16-24. No better scheme to work with the prefixes of length < 16 and > 24. The second drawback is that no prefect hashing function can be obtained in advance and thus the worst-case power consumption is still too high. In order to eliminates the drawbacks of bit selection schemes, the authors proposed another set of schemes, the trie-based partitioning schemes.

Their trie-based partitioning schemes use a two-level TCAM organization. The first-level TCAM is responsible for matching the target IP and obtaining an index of the second-level TCAM bucket. The main idea is to find a subtrie or a set of subtries and put the prefixes in the subtrie(s) in a single TCAM bucket such that all the buckets are balanced. Two different split algorithms were proposed. They are *subtree-split* and *postorder-split* algorithms. Given a parameter b, the subtree-split algorithm can partition the routing table into K buckets, where K $\in [\lceil N/b \rceil, \lceil 2N/b \rceil]$ and each bucket contains $\lceil b/2 \rceil$ to b prefixes. In addition, an index TCAM of size K and an index SRAM are needed. The drawbacks of the subtree-split algorithm are as follows. Usually, K is fixed in advance. There is no simple and efficient method to estimate how big is b for obtaining the expected K. Also, the numbers of prefixes stored in TCAM buckets are not balanced; the worst-case difference between the numbers of prefixes in two TCAM buckets is $\lceil b/2 \rceil$. The worst is that the subtree-split algorithm may fail when K is fixed and TCAM memory pressure to store all the prefixes is high. Another drawback is that the subtree-split algorithm does not provide an incremental update when a new prefix is going to be inserted into a full TCAM bucket. For this situation, only re-partitioning is the solution. The postorder-split algorithm remedies most of the drawbacks of the subtree-split algorithm. It partitions the routing table into K buckets each containing exactly $\lceil N/K \rceil$ prefixes, except possibly the last bucket. However, the postorder-split algorithm comes at the cost of large index TCAM, especially when K is large. The postorder-split algorithm can move the prefixes in an overflowed TCAM bucket to a neighboring bucket. However, it is not clear that both neighboring buckets of an overflowed bucket can be used as the buffering space to hold overflowed prefixes. Also, when both neighboring buckets of an overflowed bucket are full, no incremental update can be achieved.

Based on their performance evaluations in terms of power consumption reduction, subtree-split performs better than postorder-split when the number of buckets grows beyond 64. On the contrary, postorder-split performs better than subtree-split when the number of buckets is less than 64.

In this paper, we use the prefix comparison technique proposed in [6] to sort the prefixes. With the sorted prefixes, we can easily partition the prefixes into K groups by selecting K - 1 pivot prefixes. Thus, all the prefixes can be compared in parallel with the target IP address to determine which group that this target IP belongs to.

Before going on, we make the following assumptions used in the paper. Similar to the assumption used in [4], we assume that an original TCAM can be divided into K buckets. An additional control line is used to select which TCAM bucket is only activated for searching the input IP. Also, we assume that the same prefix can be distributed over many buckets. When a prefix is to be deleted, we assume that we can select a TCAM bucket or all the TCAM buckets from which the prefix is deleted.

3 Proposed Partitioning Algorithms

It is known that if a list of data items can be sorted then they can be easily partitioned into groups such that all the data items in one group are smaller/larger than all items in another group. Next, a list of pivot items that are the largest or smallest items in all the groups can be selected as the first level data items. As a result, the search process for a data is operated in two steps. The first step searches the first level pivot list and determines which group the searched data belongs to. The second step then searches the determined group for the data. Before we can apply the same technique to reduce power consumption in TCAM, we need a comparison mechanism to sort the prefixes in routing tables and partition them into groups. Comparing prefixes is not easy because the lengths of the prefixes may be different. In [6], we have proposed a systematic method to compare prefixes of various lengths. Based on the proposed comparison mechanism for the prefixes of different lengths, the proposed prefix partitioning algorithm can be developed.

In order to devise a new partitioning technique based on the prefix comparison, we also need to consider the enclosure property among the prefixes since shorter prefixes may cover more than one group. The enclosure prefixes must be put into the groups that contain longer prefixes covered by them. Therefore, we need to duplicate the enclosure prefixes and put them in both groups. In summary, the problem of partitioning the prefixes into K groups becomes (1) how to select the pivot prefixes to cut the tree evenly into groups, (2) which enclosure prefixes are duplicated, and (3) which groups to put the duplicated prefixes. Before describing the detailed design of the proposed algorithm, we give the definition of prefix comparisons as follows.

Definition 1 of prefix comparison: The inequality $0 < * < 1$ is used to compare two prefixes in the ternary representation of prefixes.

Instead of sorting the prefixes in a routing table by a fast sorting algorithm such as quick sort, we use the binary trie to complete the sort operation. The binary trie will also be used for determining the enclosure prefixes. Given a routing table, the binary trie is first constructed. Then we perform an inorder traversal in the binary trie and obtain the sorted prefixes. Assume there are N prefixes in the routing table and we want to partition them into K groups. We simply divide the sorted prefixes evenly into K groups. Each group contains $\lceil N/K \rceil$ prefixes. We select the largest prefix in each group as the pivot, except the last group. There are $K - 1$ pivot prefixes, *pivot[i]* for i=0 … $K - 2$.

Now we describe how to determine which enclosure prefixes should be duplicated and inserted into which group. For a group i, we consider the largest and smallest prefixes L_i and S_i. We first traverse the binary trie bottom-up from L_i to the root. If a valid enclosure prefix, P_{enc}, that encloses L_i is found and P_{enc}

$>$ pivot[i] or $P_{enc} \leq$ pivot[i−1], then P_{enc} is duplicated and put into group i. Secondly, similar traversal from S_i to the root can be done. The two traversals for L_i of group i and S_{i+1} of group i+1 can be combined into one by performing the traversal for the longest common ancestor of L_i and S_{i+1}. This is because no valid prefix exists between L_i and S_{i+1} and the longest common ancestor of L_i and S_{i+1} must locate between L_i and S_{i+1}. We call the above procedure for duplicating enclosure prefixes as the *group cut procedure*. Theoretically, each group cut generates at most W extra prefixes. We present the detailed algorithm for the proposed prefix partition as follows.

```
Prefix-partition(K, RoutingTable)
{
    Construct the binary trie (BinaryTrie) from the routing table (RoutingTable)
    Inorder traverse BinaryTrie and store all valid prefixes
        in array Plist in sorted order.
    Select K − 1 prefixes, pivot[i] for i = 0 .. $K-2$ such that
        pivot[i] = Plist[$i*b-1$], where b= ⌈N/K⌉.
    For (i = 0; $i < K-1$; i++) {
        Put prefixes, Plist[i*b] to Plist[i*b + b − 1] in group i.
        For each of the ancestor prefixes (P) of the longest common
            ancestor of Plist[i*b + b − 1] and Plist[i*b + b] do
                If P is not in group i then put P in group i.
                If P is not in group i + 1 then put P in group i + 1.
    }
    Put prefixes Plist[K*b − b] to Plist[size(Plist) − 1] in group K − 1.
}
```

By using ⌈N/K⌉ to be the number of prefixes in each group, we may underestimate the impact of duplications of enclosure prefixes. Therefore, in real implementation, we can use a kind of greedy algorithm by increasing the size of ⌈N/K⌉ to balance the number of prefixes in each TCAM bucket. However, based on our experiments, the greedy algorithm produces no bigger difference. Also in order to make insertion process simple, the pivot prefixes are selected in such a way that they are disjoint with each other. The selected pivot prefixes are restricted not to be that of length 32. This restriction can be fulfilled by selecting a pivot prefix between groups i and i + 1 that is larger than the largest prefix in group i and smaller than the smallest prefix in group i + 1. This restriction for selecting a pivot prefix is for using 32 bits to represent a prefix and will be explained later. Based on the above analysis, we have the following result.

Theorem 1: The size of each group created by the prefix partitioning algorithm is at most ⌈N/K⌉ + W, where the size of the routing table is N and maximum length of prefixes in the routing table is W.

The architecture of the proposed prefix partitioning algorithm is illustrated in Figure 1. Each partitioned group of prefixes is in fact treated as a TCAM bucket. The total power consumption for each lookup is equal to the power consumed by the selected TCAM bucket and the first level circuitry. As shown

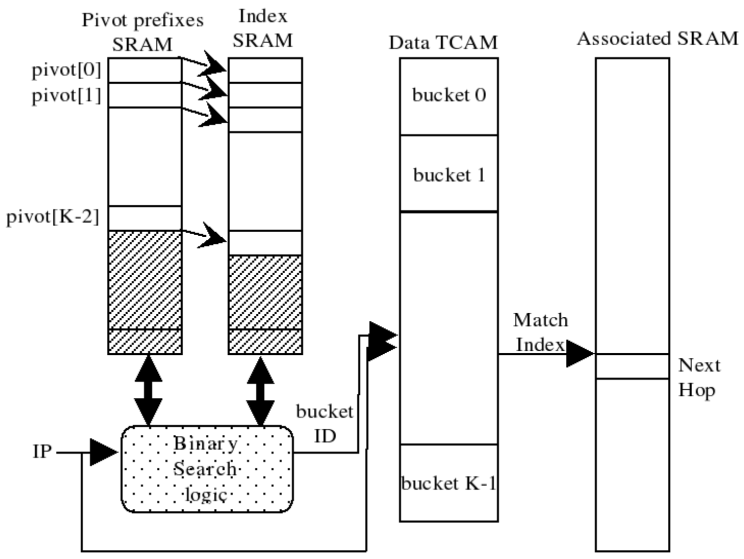

Fig. 1. TCAM architecture for the proposed prefix partitioning algorithm.

in Figure 1, the binary search logic first lookups the pivot array and obtains the TCAM bucket number from the associated index array. Then the TCAM bucket number is input in data TCAM and only that TCAM bucket is looked up.

Route Update. The route update consists of two steps. The first step is to determine if the inserted prefix needs to be duplicated and which groups to insert. The second step is to avoid recomputed the prefix partition process because re-partitioning involves excessive TCAM write operations.

Recall that the prefixes in the pivot array are disjoint and are in an increasing order. When inserting a prefix P, we perform the following steps. First we use the binary search to locate the position at which P is supposed to reside. Assume that pivot[i−1] < P < pivot[i]. We consider two conditions. The first condition is when P is disjoint with both pivot[i−1] and pivot[i] or P is enclosed by either pivot[i−1] or pivot[i], but not both. The second condition is when prefix P encloses one or more pivot prefixes. If the first condition is met, prefix P is inserted into group i since no other group will be affected. If the second condition is met, we need to search for a set of successive pivot prefixes that are enclosed by prefix P. Then we duplicate prefix P and put a copy of P in each TCAM bucket that is enclosed by P. The deletion of an old prefix requires only a delete command issued to all the TCAM buckets to remove the prefix if it exists.

The TCAM partitioning algorithms proposed in [4] assumed that TCAM buckets need to be re-partitioned each time any bucket overflows. Therefore, the frequency of re-partitioning depends on two factors. One is the route addition or deletion rate. The other factor is the prefix occupation pressure in each TCAM bucket. As shown in [4], if we have a low prefix occupation pressure in any

TCAM bucket, the number of times that a re-partitioning process is needed will be very small and thus the updating impact will be negligible. In this section, we shall propose an update scheme that avoids re-partitioning even when the TCAM bucket to be inserted is full.

Assume a prefix P is supposed to be inserted in TCAM bucket i. We first consider the case when TCAM bucket i is full and TCAM bucket i + 1 is not full. Let L_i and M_i be the largest and the second largest prefix in TCAM bucket i. We select another pivot Q_i such that $M_i \leq Q_i < L_i$. Next, we move L_i to group i + 1 and let pivot[i] = Q_i. Similar operations can be performed when TCAM bucket i − 1 is not full. Notice that the smallest bucket (bucket 0) or largest bucket (bucket K) has only one neighboring bucket instead of two. Assume that L_i and S_i are the largest and smallest prefixes in bucket i, respectively. By using S_0 and L_K as additional pivot prefixes, buckets 0 and K can have two neighboring buckets and thus less chance to perform the routing table re-partitioning.

As stated in [4], there are occasional floods of up to a few thousand route additions in one second in the real update router traces. Also, these route additions are very close to each other in a subtrie of the binary trie built from the routing table. These routes are often subsequently withdrawn. These floods often cause a single bucket or even three consecutive buckets to repeatedly overflow. Therefore, we may need a better scheme to avoid repartitioning the routing table even when buckets i − 1, i, and i + 1 are full. The proposed prefix update algorithm is as follows. Assume the new routing prefixes need to be inserted in bucket i. Firstly, we select a new pivot prefix Q_i in bucket i such that is $M_i \leq Q_i < L_i$, where L_i is the largest and M_i is the second largest in bucket i. Secondly we select a non-full bucket j. The prefix L_i is moved to bucket j. Bucket j can be selected to be the one containing the smallest number of prefixes at the time of insertion. Let P_i = pivot[i]. Thirdly, we assign pivot[i] = Q_i and add P_i as an extra pivot between pivot[i] and pivot[i + 1]. Notice that the pivot[j] for j = i + 1 to K − 2 must be shifted to allocate an empty entry to put P_i. The corresponding entry in index SRAM must also be allocated by shift operations and set this index TCAM entry to point to bucket j.

32-bit prefix representation. The two most important operations needed in the IP lookup are matching a prefix with a 32-bit IP address and comparing two prefixes of various lengths. The matching operations have already been taken care by the hardware logics in TCAM. The prefix comparison involves ternary operations where 0, 1, and the don't care bit (denoted as *) are checked. In [6], we proposed a method to represent the prefixes in a binary format using 33 bits in the context of IPv4. The method can be straightforwardly expanded to IPv6. We formally define the binary representation of a prefix as follows.
Definition 2 of (n+1)-bit representation for the prefixes in an n-bit address space: For a prefix of length i, $b_{n-1}b_{n-2}b_{n-i}$, where b_j=0 or 1 for $n - i > j \geq 0$, its binary representation is $b_{n-1}b_{n-2} \ldots b_{n-i}1a_{n-i-1} \ldots a_0$, where a_j=0 for $j = 0 \ldots n-i-1$.*
Since current processors usually have 32-bit wide instructions. The 33-bit prefix representation thus needs two 32-bit storages. A naive implementation of the

comparison operations for the 33-bit representation can lead to inefficiency under 32-bit instructions. Two 32-bit binary comparison operations may be needed in the worst case since only 32-bit arithmetic and logic operations are available in current 32-bit processors. It also means that two 32-bit memory reads are needed if the size of registers is 32 bits.

If there is no prefix of length 32, we can use only 32 bits by ignoring the least significant bit that must always be zero. This is the case for computing the pivot prefixes in the proposed prefix partition algorithm. Note that a pivot is the common ancestor of two prefixes and thus has a length of at most 31. Therefore, when we use 32 bits to represent a pivot prefix of length less than 32, the 33^{th} bit of a pivot prefix must be zero. However, if we use 32 bits to represent a prefix of length 32 (in fact, an IP address) the 33^{th} bit of the prefix is assumed to be one. Therefore, when we compare an IP with a pivot prefix using 32-bit representation, equality means the IP is larger than the pivot prefix. Therefore, the 32-bit representation for a 32-bit ternary prefix is sufficient.

Table 1: general information of the routing tables used in the paper.

	Funet-40k	Oix-80k	Oix-120k	Oix-150k
# of prefixes	41,709	89,088	120,635	159,930
Length distr.	8-30-32	6-32	8-32	8-32
Date	1997-10-30	2003-12-28	2002-12-01	2004-2-1

4 Performance Evaluations

In this section, we present the performance results for the subtree-split and postorder-split algorithms [4] and the proposed T-bit expanded and prefix partitions. Table 1 shows the general information of the routing tables used in the paper. Three routing tables are the snapshots of the routing table traces obtained from University of Oregon Route Views Archive Project [oix]. The fourth table, i.e., the smallest among the four routing tables, is obtained from the web site provided by the authors of the LC trie [3]. For example, Funet-40k routing table contains only prefixes of length 8 to 30 and 31. With these routing tables of various sizes, we can obtain complete understanding on the performance when applying different TCAM architectures. We take K, the total number of TCAM buckets, as the input to the algorithms. The output will be the reduction in *power consumptions* which is defined as *the ratio of the total number of routing table prefixes to the maximum number of prefixes searched in all TCAM buckets for an IP lookup.*

For subtree-split and postorder-split algorithms and the proposed prefix partition, the maximum number of prefixes searched is the sum of the total number of routing entries in the index TCAM or in the pivot engine and the number of prefixes in the largest bucket in the data TCAM. As for the index SRAM and associated SRAM with routing information, its power consumption is small and can be ignored in the power consumption calculation.

The reductions in power consumption for four routing tables are shown in Figure 2. The proposed prefix partition performs consistently better than sub-

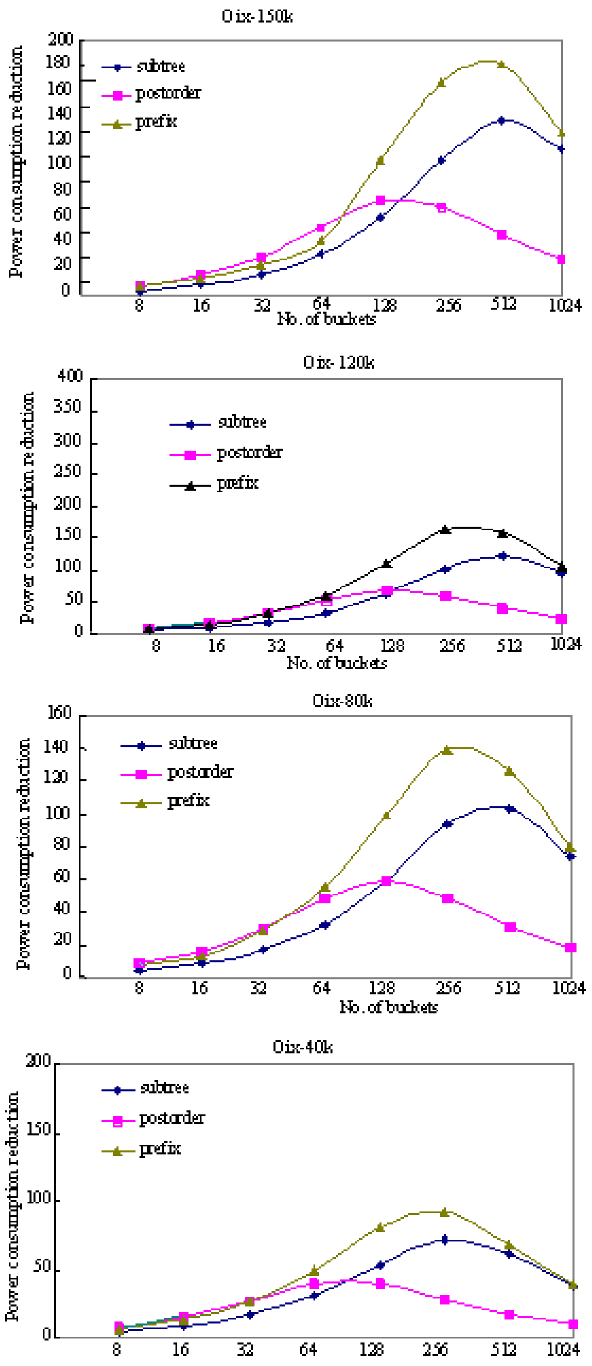

Fig. 2. Power consumption reduction for subtree-split, postorder-split, and prefix-partition algorithms.

tree split for all the routing tables with various sizes. When the number of buckets K is less than 256, the proposed prefix partition has 33% to 91% more power consumption reduction than subtree split. When K increases up to 1024, the difference between the prefix partition and subtree split is minimal because the size of index TCAM begins to dominate.

By carefully inspecting the results of the postorder split and the proposed prefix partition, they both have the same characteristics that the reduction drops when the number of buckets K becomes larger. This is because the size of index TCAM dominates for postorder split and duplicated prefixes of shorter lengths increases in each TCAM bucket for the proposed prefix partition. Their performance stays very closely when K is less than 32. However, the power consumption reduction of postorder split drops much more than that of the prefix partition when K increases.

5 Conclusion

In this paper, we introduced a new partitioning architecture and algorithm to reduce the TCAM power consumption. The prefix partition is shown to perform better than the existing subtree-split and postorder-split schemes. Prefix partition scheme has the similar characteristics to the subtree-split in power consumption reduction. However, prefix partitioning algorithm performs consistently better than subtree-split. One major advantage of the prefix partitioning algorithm is that it is capable of performing incremental updates.

References

1. M. A. Ruiz-Sanchez, Ernst W. Biersack, and W. Dabbous, "Survey and taxonomy of IP address lookup algorithms", IEEE Network Magazine, 15(2):8–23, March/April 2001.
2. D. Meyer, "University of Oregon Route Views Archive Project" at `http://archive.routeviews.org/`.
3. S. Nilsson and G. Karlsson "IP-Address Lookup Using LC-Tries", IEEE Journal on selected Areas in Communications, 17(6):1083-1092, June 1999.
4. F. Zane, G. Narlikar, and A. Basu, "CoolCAMs: Power-Efficient TCAMs for Forwarding Engines," in Proc. INFOCOM 03, March 2003.
5. D. Shah and P. Gupta, "Fast Updating Algorithms for TCAMs", IEEE Micro, pp.36-47, 2001.
6. Y. Chang, "Simple and Fast Binary Prefix Searches for IP Lookups", submitted for publication.