An Efficient IP Lookup Architecture with Fast Update Using Single-Match TCAMs*

Jinsoo Kim and Junghwan Kim**

Department of Computer Science, Konkuk University,
322 Danwol-dong, Chungju-si, Chungbuk 380-701, Korea
{jinsoo,jhkim}@kku.ac.kr

Abstract. The increasing demand for new multimedia services requires the higher performance routers. The performance of Internet router highly depends on the efficiency of update operations as well as lookup operations on IP forwarding table. While IP lookup schemes based on TCAM(Ternary Content Addressable Memory) can achieve high speed lookup, they usually need more complex update operations because of the ordering constraint on prefixes. In this paper we propose an efficient IP lookup architecture to provide fast update using a new type of TCAM named single-match TCAM. Also, we present elaborated algorithms to guarantee that each single-match TCAM generates at most one match for a given destination IP address. In our scheme the updating overhead can be reduced because there is no ordering constraint on the single-match TCAM. We evaluate as well the update performance of our scheme through simulation under real forwarding tables and update data.

Keywords: Internet router, single-match TCAM, IP lookup, fast update.

1 Introduction

The qualities of wireless/mobile services as well as wired services highly rely on the performance of the Internet. A diversity of multimedia applications including mobile services has been explosively invented and the number of hosts and users has increased on the Internet. Accordingly, Internet traffic has exponentially increased as well. To guarantee the service qualities under the growing Internet traffic, it is necessary to improve remarkably the performance of the router which is a key element in the Internet.

IP address lookup is one of the most important functions in Internet routers. In order that a router forwards an incoming packet to its final destination, the router must determine the output port or the next hop address by looking up the matching prefix in the forwarding table based on the destination address of the packet. The IP lookup operation becomes more and more computationally intensive because the variable-sized prefixes have been extensively employed since the advent of CIDR(Classless Inter-Domain Routing). Since several prefixes can be matched for a destination IP

* This work was supported by Konkuk University.
** Corresponding author.

J. Harju et al. (Eds.): WWIC 2008, LNCS 5031, pp. 104–114, 2008.
© Springer-Verlag Berlin Heidelberg 2008

address under the CIDR, the router has the burden to select the longest matching prefix(LMP) as the best matching one. Therefore, the performance of the router strongly depends on the efficiency of the IP lookup operation.

Many researchers have studied fast lookup schemes for the development of the high performance routers[1, 2]. Most of the schemes can be classified into software approaches based on trie and hardware approaches based on TCAM(Ternary Content Addressable Memory). Trie-based IP lookup schemes usually require several memory accesses per lookup and those accesses may be serialized. In contrast, TCAM can perform a lookup operation in a single cycle owing to its parallel access characteristics. Therefore, TCAM have been paid much attention to in recent years.

TCAM is a fully associative memory in which each memory cell can store a "don't care" state in addition to 0's and 1's states. Thus, TCAM can look up variable-length prefixes for a given destination address. Because there may be several matches in an IP lookup operation, it is required to determine the best match, *i.e.*, LMP. For the determination of the LMP, all prefixes of a TCAM needs to be ordered by some criteria such as length, the ancestor-descendent relationship and level on the prefix search trie. Under the ordered circumstance a priority encoder can select the LMP on the uppermost location among all matched prefixes.

The forwarding table in a router is frequently updated to avoid Internet instability[3]. In particular, Internet backbone router should be able to deal with a few hundred or thousand updates per second. Most of TCAM-based lookup schemes may experience several movements of prefix entries for a single update because the ordering must be maintained in the TCAMs. Therefore, frequent updates may consume many computation cycles in the IP lookup engine and result in the degradation of the lookup performance. The efficient update is one of the most important issues together with lookup performance and power management in TCAM-based schemes.

In this paper, we present a new architecture to provide fast update by using single-match TCAMs. Our elaborated algorithms guarantee that each single-match TCAM generates at most one match for a given destination address. So, it can eliminate both the ordering constraint and the priority encoder in a single-match TCAM, which makes the update fast. The rest of this paper is organized as follows. Related works on fast update of TCAM are described in section 2. In section 3 we propose our IP lookup architecture for fast update and describe the functionality of each component. In section 4 we present the algorithms for IP lookup, insertion and deletion of a prefix respectively. The performance of the proposed scheme is evaluated in section 5. Finally, we conclude this paper in section 6.

2 Related Works

Several methods have been proposed to reduce the updating overhead. Shah and Gupta[4] proposed the two fast updating algorithms which are PLO_OPT and CAO_OPT, to reduce the number of memory movements during update. In PLO_OPT all the existing prefixes are sorted by their lengths and free locations are reserved in the middle of the table. Then the number of memory movements per update is no more than $L/2$ where L is the maximum prefix length, *i.e.*, 32 in IPv4. CAO_OPT exploits the fact that the ordering needs to be maintained only between two prefixes one of which is the prefix of the other. In this algorithm the ordering is

referred to as the chain-ancestor ordering(CAO) where a chain means the collection of the prefixes on the path from the root to a leaf node in a prefix search trie. In CAO_OPT the worst case number of memory movements per update has been reduced to $D/2$ where D is the maximum length of chains.

Wu et al.[5] presented an update algorithm based on the prefix level. A forwarding table is divided into several partitions according to the levels in a prefix search trie and the partitions are ordered by its level. Each partition includes free space for future update. A new prefix can be easily inserted if its parent and children are in different level partition. However, the free space may contain prefixes of different levels within the free space as update proceeds by means of that algorithm. It will cause the memory movements as in the CAO_OPT. In case of deletion it leaves the deleted space as unavailable, so it wastes memory severely. Moreover, it is difficult to predetermine the size of each free space because the distribution of updates cannot be estimated in advance.

Several approaches have been researched to remove both the ordering constraint and the priority encoder module. Kobayashi et al.[6] modified TCAM by adding vertical OR circuits in the mask column to directly select the longest mask among the matched entries without priority encoder logic. It does not require any ordering constraint, so fast update can be achieved. Actual lookup delay may be increased due to the vertical ORing, even though the delay time can be reduced by pipelining technique. Ng and Lee[7] partitioned a forwarding table into several TCAM modules so that each module only contains prefixes with the same output port number. A new prefix can be inserted at any location within a TCAM module without considering the ordering. However, many updates just change the output port numbers of the existing prefixes. Such updates lead to excessive memory movements because the prefix must be moved to another module associated with new output port.

3 Proposed IP Lookup Architecture

3.1 Conventional TCAM-Based Architecture

Conventional TCAM-based IP lookup architecture consists of a TCAM, a conventional data memory and a priority encoder as shown in Fig. 1. For a given destination IP address, there are possibly multiple matches in the TCAM and the priority encoder selects one final matched entry among those matches. The entry in the data memory which corresponds to the final matched entry of the TCAM contains target output port number. Using the output port number the packet can be delivered to the target port.

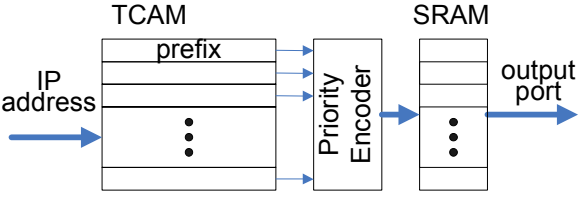

Fig. 1. Conventional TCAM-based IP Lookup Architecture

Since the priority encoder selects the final match among several matches by means of their location, the prefixes in the TCAM should be ordered so that the longest prefix always locates prior to the other matched prefixes.

For example, let's consider the sample forwarding table with 10 prefixes as shown in Fig. 2. Those prefixes can be stored in partial order of p_1, {p_2, p_3}, {p_4, p_5, p_6}, {p_7, p_8, p_9}, p_{10}, in which prefix entries within a brace can be located in any order. If an 8-bit destination address 10100100 is given, then two prefixes p_1=10100* and p_8=10* are matched. Because the prefix p_1 is located prior to the prefix p_8 in the TCAM, the priority encoder can select the prefix p_1 as the LMP. The output port 10 of the corresponding entry in SRAM can be found.

Entry	Prefix	Length	Port	Entry	Prefix	Length	Port
p_1	10100*	5	10	P_6	110*	3	14
p_2	1011*	4	11	P_7	00*	2	15
p_3	1110*	4	11	P_8	10*	2	16
P_4	010*	3	13	P_9	11*	2	17
P_5	100*	3	14	p_{10}	0*	1	18

Fig. 2. A Simple Example of Forwarding Table

3.2 Design of the Proposed Architecture

The maximum number of matched entries in a TCAM depends on the maximum depth of levels of the prefix search trie. For a given destination IP address, the prefixes which have ancestor-descendant relation will be matched simultaneously. The depth of a prefix search trie currently does not exceed 7 even including the default prefix so there can be at most 7 matches. If the forwarding table is partitioned into several TCAMs so that there is no ancestor-descendant relation in each partitioned TCAM, then it is guaranteed that there exists at most one match in each TCAM. It means that the TCAMs do not need a priority encoder any more. In section 4 we will describe how to satisfy such single-match condition when prefixes are inserted to TCAMs.

Fig. 3 shows our proposed architecture for IP lookup. It consists of 8 partitioned TCAMs and selection logic. Each of TCAMs has supplementary SRAMs which contain the lengths of prefixes and output port numbers. Note that $TCAM_0$ to $TCAM_6$ don't have any priority encoder logic whereas the last $TCAM_7$ has the priority encoder. The $TCAM_7$ is similar to the TCAM with a priority encoder used in conventional IP lookup architecture.

3.3 Single-Match TCAMs

In order to describe our single-match TCAMs, we define the terminologies which are *disjoint* and *disjoint set* as follows. Those are similarly defined in several literatures including [8].

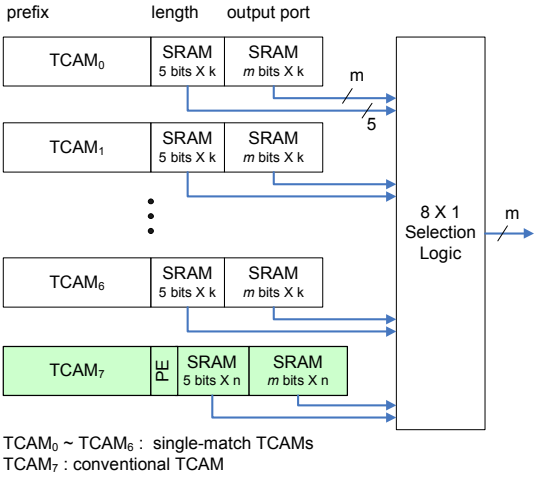

Fig. 3. Proposed IP Lookup Architecture

Definition 1. Two prefixes p_i and p_j are *disjoint* if neither p_i is the prefix of p_j nor p_j is the prefix of p_i.

For example, p_1=10100* and p_2 = 1011* of Fig. 2 are disjoint because neither p_2 can be a prefix of p_1, nor p_1 can be a prefix of p_2. However, p_8 = 10* and p_2 are not disjoint because p_8 is a prefix of p_2.

Definition 2. A set of prefixes P is the *disjoint set* if any two $p_i, p_j \in P$ are disjoint.

Each of $TCAM_0$ to $TCAM_6$ should contain a disjoint set of prefixes, *i.e.*, any prefixes in each TCAM are disjoint each other. So the result of lookup for a given IP address will be no more than one match in each TCAM. We call such TCAMs "*single-match*" TCAMs. Obviously, the single-match TCAMs don't have priority encoder logic.

The prefixes in a forwarding table are partitioned into several disjoint sets and each set is mapped onto a single-match TCAM. There is at most one match in each TCAM. Although our architecture has a conventional TCAM, the conventional TCAM will generate one final match through the priority encoder. The selection logic selects longest one among those matches by using length data. The selection logic finally sends out the corresponding output port number.

Since any prefixes which are in ancestor-descendant relation in a prefix search trie are certainly not disjoint, the depth of a prefix search trie is the minimum number of disjoint sets which is sufficient to be the number of single-match TCAMs needed. However, we need a conventional TCAM additionally in our architecture except the single-match TCAMs. The rationale for the conventional TCAM is related to the fact that the disjoint sets will be varied through several updates and it is hard to re-map those sets into single-match TCAMs by online algorithm without burden. Each single-match TCAM may be required to move the existing prefixes to another single-match TCAM in order to maintain a disjoint set when it inserts a new prefix. For example, suppose that there are only two single-match TCAMs and two disjoint prefixes

p_1=10100* and p_2 = 1011* are stored in different single-match TCAMs, then there is no way to insert a new prefix, p_8=10* into any of the single-match TCAMs without moving an existing prefix. Movements of existing prefixes are not desirable for fast updating, so we resolve the problem by providing an additional TCAM which is a conventional TCAM with a priority encoder. In case that there is no suitable single-match TCAM for a new inserting prefix, the conventional TCAM will be assigned. Since the conventional TCAM has a priority encoder, any new prefix can be inserted regardless of whether it is disjoint with the existing prefixes.

While the prefixes of the conventional TCAM need to be ordered, the prefixes of the single-match TCAMs do not. So, both insertion and deletion can be performed faster in the single-match TCAMs than in the conventional one. In section 5, the experiment result shows that most of prefixes are stored in the single-match TCAMs and very small amount of prefixes are in the conventional TCAMs.

4 IP Lookup and Update Algorithms

In this section, we describe an IP lookup algorithm to search for the LMP and also present update algorithms for inserting a new prefix into an appropriate TCAM and deleting a prefix.

4.1 Search Algorithm

Fig. 4 shows the algorithm to search for the LMP with a destination IP address, *ip_addr* as a key. Each TCAM independently performs line 2. In line 2, at most one matching prefix is found using the function match($TCAM_i$, *ip_addr*). In case that there is no matching prefix in $TCAM_i$, *entry*[*i*].*length* becomes 0. Line 4 is performed by the selection logic. After it selects the TCAM containing the LMP, the corresponding output port number will be returned (line 5).

```
Search(ip_addr: an ip address)
1.   for i ← 0 to 7 do in parallel
2.      entry[i] ← match(TCAM_i, ip_addr)
3.   endfor
4.   Find k such that entry[k].length is the largest one
           among  entry[i].length for all 0≤i≤7
5.   return entry[k].output_port
```

Fig. 4. Search Algorithm for LMP

For example, the prefixes of Fig. 2 can constitute disjoint sets for $TCAM_0$ to $TCAM_6$ which are $\{p_1, p_7\}$, $\{p_2, p_4\}$, $\{p_3\}$, $\{p_5\}$, $\{p_6\}$, $\{p_{10}, p_8\}$, $\{p_9\}$, respectively. If an 8-bit destination address 10100100 is given, then only $TCAM_0$ and $TCAM_5$ contain the matched prefixes p_1=10100* and p_8=10*, respectively. So, *entry*[0] and *entry*[5] are set by the location of p_1 in $TCAM_0$ and that of p_8 in $TCAM_5$, respectively. Unless conventional $TCAM_7$ contains the matching prefix whose length is longer than 5 of p_1's length, the algorithm in Fig. 4 returns the value 10 of *entry*[0].*output_port*

4.2 Insertion Algorithm

Fig. 5 describes an algorithm to insert a new prefix p to a forwarding table. As shown in lines 1 to 6, it finds out all available single-match TCAMs to which the new prefix can be inserted. Such a TCAM satisfies that the new prefix p and every prefix ep in the TCAM should be disjoint and there should be at least one free slot in the TCAM (lines 3).

```
Insert(p: a prefix)
1.   for i ← 0 to 6 do in parallel
2.      Initialize available[i] ← false
3.      if ((p and ep are disjoint, ∀prefix ep∈TCAM_i)&&
           (there is some free space in TCAM_i))
4.           available[i] ← true
5.      endif
6.   endfor
7.   if (available[i] = false, ∀i 0≤i≤6)
8.      insert_to_tcam(p, TCAM_7)
9.   else
10.     k is randomly selected from available[i]
11.     insert_to_stcam(p, TCAM_k)
12.  endif
```

Fig. 5. Insertion Algorithm

For example, assume that the new prefix 101* is inserted under the same conditions as the example of section 4.1. Then, the prefix 101* isn't disjoint with p_1=10100*, p_2=1011* and p_8=10* which are in $TCAM_0$, $TCAM_1$, and $TCAM_5$, respectively. Consequently, *available*[*0*], *available*[*1*] and *available*[*5*] keep the *false* value. On the other hand, every prefix in $TCAM_2$, $TCAM_3$, $TCAM_4$ and $TCAM_6$ is disjoint with the new prefix 101*. Therefore, one of these TCAMs is randomly selected for insertion, if it has free space. As another example, assuming the new prefix is 1*, the prefix 1* is not disjoint with at least one prefix in each of $TCAM_0$, to $TCAM_6$. and the prefix should be inserted in $TCAM_7$.

It is easily determined if there is any non-disjoint prefix in a TCAM with respect to the new prefix. Given a 32-bit IP address TCAM searches for matched prefix in a normal lookup operation. We can give the TCAM a prefix as a search key instead of a full 32-bit IP address. If there is any non-disjoint prefix in the TCAM with respect to the prefix, match will occur in that operation. There is a simple technique to regard a prefix as a search key by considering the remaining bits of the prefix in 32-bit representation as don't care conditions[8].

If there is no available single-match TCAM in line 7 of Fig. 5, the new prefix must be inserted into $TCAM_7$ which is the conventional TCAM. Otherwise, the new prefix can be inserted into a TCAM randomly chosen from the available single-match TCAMs. The function insert_to_tcam() inserts the prefix p to the conventional TCAM with satisfying the ordering constraint. That functionality can be implemented by

applying one of various update algorithms for the conventional TCAM. The insert_to_stcam() can simply inserts the prefix p into any free location in a single-match TCAM irrespective of the ordering.

4.3 Deletion Algorithm

The algorithm to delete a prefix p from the forwarding table is shown in Fig. 6. For the prefix deletion it is needed to determine which TCAM contains the prefix p as shown in line 1. Then the prefix can be deleted from the TCAM (line 2). Actually the both steps (lines 1 and 2) can be performed together. The function delete_from() is performed differently whether it operates on conventional TCAM or single-match TCAM.

We manage contiguous free space for single-match TCAM, which causes one memory movement on deletion. However, it does not need any other memory movements because there is no ordering constraint on single-match TCAM. In case of conventional TCAM the number of memory movements differs according to update algorithms. Assuming free space is contiguous in the conventional TCAM, it also requires at least one memory movement on deletion.

```
Delete(p: a prefix)
1.  Find k such that p ∈ TCAM_k
2.  delete_from(p, TCAM_k)
```

Fig. 6. Deletion Algorithm

5 Performance Evaluation

5.1 Simulation Environment

In our simulation we used routing tables from Route Views[9]. The update data streams for 4 weeks were used for the experiment where the data of each week were separately taken from different months. For experiment we needed to convert the routing tables into forwarding tables and filter the update data streams for the forwarding tables. Table 1 shows statistics on the forwarding table and the update data streams.

Table 1. Statistics of Sampling Data

	Jul 2007	Aug 2007	Sep 2007	Oct 2007
No. of Prefixes	243511	244095	242635	248389
No. of Updates	164467	527204	787944	651771

The number of updates only includes the number of insertions and deletions but not that of modifications of output port number. We evaluate the updating performance just by the number of memory movements incurred by updates, but the modification of the output port does not cause the memory movements.

Table 2 shows the number of memory movements per update in various updating schemes. Since there is no ordering constraint on single-match TCAM, the actual number of memory movements incurred by each update may be 0. However, we adopted a policy that free space of TCAM should be contiguous, so any deleted space needs to be compacted to the contiguous free space. It causes almost one memory movement per deletion on the average. In Table 2 the column represented by sTCAM shows the number of memory movements in single-match TCAM. The other columns show those for various updating schemes in conventional TCAM, which are summarized in [4].

Table 2. Comparison of Memory Movements

Memory Movements	sTCAM	TCAM		
		L-algorithm	PLO_OPT	CAO_OPT
Insertion	0	7.27	4.1	1.02
Deletion	1			

5.2 Simulation Results

Table 3 shows the average number of memory movements per update in our scheme. Each prefix in the forwarding table is randomly assigned to a single-match TCAM unless that prefix is not disjoint with any prefix in the TCAM. If an update occurs in single-match TCAM and the update is deletion, then the number of memory movement is calculated as one. But, if the update is insertion, there is no memory movement in single-match TCAM. In case of conventional TCAM, CAO_OPT was applied to evaluating the number of memory movements. Note that the effective update cost is cheaper in single-match TCAM than in conventional TCAM. In our scheme the average number of memory movements per update ranges from 0.4902 to 0.5061, which is half as large as CAO_OPT.

Table 3. Memory Movements per Update

Mem. Movements	Jul 2007	Aug 2007	Sep 2007	Oct 2007
Moves/Update	0.4902	0.5061	0.4954	0.4965

Fig. 7 shows the number of prefixes initially contained in each TCAM. The single-match TCAM is randomly chosen, so the prefixes can be evenly distributed among the single-match TCAMs. The number of prefixes in conventional TCAM is very small and merely 821, which is 0.33% of total prefixes. It implies that the conventional TCAM which requires the ordering constraint can be constructed as very small size in our architecture.

The good updating performance is due to the fact that most of updates are performed in single-match TCAMs and very few updates are performed in conventional TCAM. Fig. 8 shows the distribution of updates over TCAMs. Most of updates concentrate on single-match TCAMs and are evenly distributed among the single-match

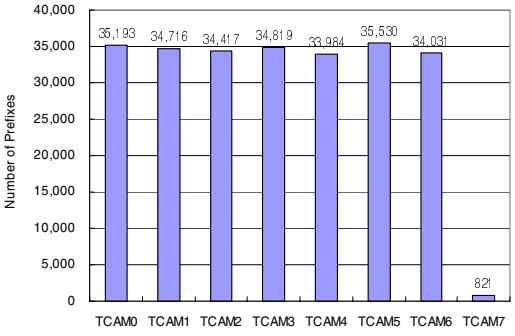

Fig. 7. The Number of Initial Prefixes in Each TCAM

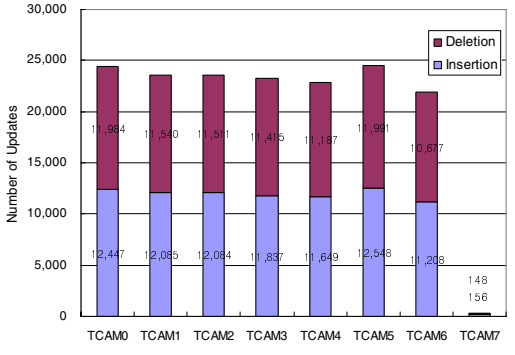

Fig. 8. Insertions and Deletions

TCAMs as well. The numbers of insertions and deletions in conventional TCAM are merely 156 and 148, which correspond to 0.19% and 0.18% of total insertions and deletions respectively. The insertion ratio, 0.19%, is much smaller than the ratio of the number of initial prefixes, 0.33%, in the conventional TCAM, which implies the relative portion of the conventional TCAM will not increase.

5.3 Discussion

The updating performance is related to two factors: the number of updates in the conventional TCAM and the number of deletions in the single-match TCAMs. In case of single-match TCAM the memory movements are only incurred by deletions while both insertions and deletions incur memory movements in the conventional TCAM.

The smaller the number of updates in the conventional TCAM is, the better updating performance is achieved. The simulation results show that the number of updates in the conventional TCAM is quite small. In the simulation a simple random assignment strategy was applied to the insertion algorithm, however, it is possible to apply various assignment strategies to the insertion algorithm. It is expected that the number of prefixes and updates in the conventional TCAM will be affected by the assignment strategies.

The number of single-match TCAMs may be controlled depending on cost and performance. It is also expected that even if the number of single-match TCAMs is reduced, the performance will not be rapidly degraded because large amount of prefixes can reside in small number of disjoint sets.

6 Conclusion

The conventional TCAM must satisfy some ordering constraint on prefixes for longest prefix matching. It results in low updating performance of the TCAM and even lookup performance would be low due to overhead of a priority encoder.

Our novel architecture shows good performance in updating by introducing a new type of TCAMs so called single-match TCAMs which do not need any priority encoder logic. By using an elaborated assignment strategy to insert a new prefix, each single-match TCAM can maintain a disjoint set of prefixes and produce at most one match. Since it does not require any ordering constraint on each single-match TCAM, a newly inserted prefix can be located at any place within a single-match TCAM. It is expected that lookup will also spend less time than in the conventional TCAM because it does not have a priority encoder.

Although the proposed architecture still requires a conventional TCAM, our simulation result shows that the number of prefixes which reside in the conventional TCAM is very small. As the result very few updates are performed in the conventional TCAM and the average number of memory movements per update is as small as about 0.5.

Novel assignment strategies for prefix insertion should be developed and evaluated in further research. The memory movements on deletions in single-match TCAM can be eliminated provided that the insertion hardware is enhanced. The design of the hardware to eliminate memory movements also remains for future work.

References

1. Ruiz-Sanchez, M.A., Biersack, E.W., Dabbous, W.: Survey and Taxonomy of IP Address Lookup Algorithms. IEEE Network 15, 8–23 (2001)
2. Chao, H.J., Liu, B.: High Performance Switches and Routers. Wiley-Interscience, Chichester (2007)
3. Labovitz, C., Malan, G.R., Jahanian, F.: Internet Routing Instability. IEEE/ACM TON 6, 515–528 (1998)
4. Shah, D., Gupta, P.: Fast Updating Algorithms for TCAMs. IEEE Micro 21, 36–47 (2001)
5. Wu, W., Shi, B., Wang, F.: Efficient location of free spaces in TCAM to improve router per-formance. IJCS 18, 363–371 (2005)
6. Kobayashi, M., Murase, T., Kuriyama, A.: A Longest Prefix Match Search Engine for Multi-Gigabit IP Processing. In: 2000 International Conf. on Communications, pp. 1360–1364. IEEE Press, New Orleans (2000)
7. Ng, E., Lee, G.: Eliminating Sorting in IP Lookup Devices using Partitioned Table. In: 16th IEEE International Conf. on Application-Specific Systems, Architecture and Processors (ASAP), pp. 119–126. IEEE Press, Greece (2005)
8. Akhbarizadeh, M.J., Nourani, M., Cantrell, C.D.: Prefix Segregation Scheme for a TCAM-Based IP Forwarding Engine. IEEE Micro 25, 48–63 (2005)
9. University of Oregon Route Views Project, `http://www.routeviews.org/`