

Reducing Routing Table Size Using Ternary-CAM

Huan Liu

Department of Electrical Engineering

Stanford University, CA 94305

huanliu@stanford.edu

Abstract

Ternary Content Addressable Memory (TCAM) has increasingly been used in high speed routers to perform routing lookup function. They allow simultaneous comparison of the key with every index at the same time so that the longest matched prefix could be selected within one memory access, much faster than software based search algorithms. We propose two techniques to compact routing table stored in TCAM. The techniques allow a smaller TCAM to be used to reduce cost, power consumption and thermal dissipation. They could also help routers to scale to larger routing table. Our simulation shows up to 48% size reduction. Fast incremental update algorithm is presented to maintain space saving without recomputing the compacted table after each routing update.

1. Introduction

Classless Inter-Domain Routing (CIDR) [1] was proposed and adopted in an attempt to slow the exhaustion of Internet Protocol (IP) address space. As a result, Internet routers need to find the longest matched prefix against destination IP address to determine how to route an incoming packet.

Various software based schemes have been proposed to speed up the lookup function[2][9][10]. However, all of these approaches require at least 4 to 6 memory accesses. With increasing high throughput requirement, the latency and bandwidth of modern memory architecture severely limit the number of memory accesses a system designer can afford. Clearly, the software based solution has a hard time to scale up to 10Gbps processing and beyond.

To cope with the problem, several hardware based solutions have been proposed. Some of the approaches[6][16] use dedicated special hardware, whereas others[4][14][15] use commercially available Content Addressable Memory (CAM).

Traditional CAM allows simultaneous comparison between all indexes and the key, and the entry

corresponding to the matched index is automatically sent to the output. The main advantage of a CAM is that the search time is bounded by one single memory access, thus it can guarantee a high lookup throughput. Regular CAM allows only fix length comparison, i.e., it is not directly suitable for varying length matching as in the case of longest prefix matching. A possible solution is to store prefix with different length into separate CAM chip, then design external logic to pick the longest matched entry from all matched CAM chip[7].

Table 1: Example prefix table stored in TCAM

#	Prefix	Mask	Next Hop port
P_1	10011100	11111100	7
P_2	10001100	11111100	7
P_3	11011100	11111100	7
P_4	10001000	11111000	5
P_5	11010000	11110000	4
P_6	11110000	11110000	7
P_7	10010000	11110000	4

Another type of CAM called Ternary Content Addressable Memory (TCAM) could be used directly to solve the longest prefix matching problem[15]. In addition to the index, it also stores a separate mask for each entry. Both the key and the indexes are masked before the comparison. An example routing prefix table stored in TCAM is shown in Table 1.

Both CAM and TCAM are commercially available[12]. However, both of them are much more costly than conventional memory, with TCAM being the most expensive. In addition, they consume more power and dissipate more heat, which poses difficult challenge for the system designer. Thus, it is desirable to compact routing table size so that a smaller number of CAM chips can be used in the system. Even if only one single CAM chip is used, a routing table compaction mechanism can help to reduce its power consumption and heat dissipation. As the number of routing prefixes is increasing steadily, a routing table compaction scheme can also help to deal with routing table size explosion. In this paper, we propose two techniques to compact routing table stored in TCAM.

2. Prefix compaction

The number of possible routes in a routing table is typically small because only a limited number of interface cards can fit into the router chassis. In contrast, the number of routing prefixes is typically very large, in the range of several thousand. For example, the number of routes and the number of routing prefixes in some backbone routers [8] at major US Internet exchange point (IXP) is shown in Table 2. The snapshot was extracted on March 7, 2001. The routing table has at least 500 times more prefixes than the number of routes. Given the disparity, there will be some structure we could exploit to reduce the routing table size. In the following, we present two such techniques.

Table 2: Routing table characteristics

	Maecast	Maewest	Pacbell	Aads	Paix
# of routes	36	40	19	37	26
# of prefixes	23554	32139	38791	15906	29195

2.1. Pruning

Pruning is a technique to eliminate redundant routing prefixes. To facilitate discussion, we first define some terms. We use $|P_a|$ to denote the length of prefix P_a , and we use $P_{a,i}$ to denote the i th bit of the prefix where $P_{a,1}$ is the most significant bit, and $P_{a,|P_a|}$ is the least significant bit. We say a prefix P_a is the *parent* of prefix P_b if the following three conditions hold:

1. $|P_a| < |P_b|$
2. $P_{a,i} = P_{b,i}$ for all $1 < i < |P_a|$
3. There are no prefix P_c such that $|P_a| < |P_c| < |P_b|$, and $P_{c,i} = P_{b,i}$ for all $1 < i < |P_c|$

Intuitively, the parent of prefix P_b is the longest prefix that matches the first few bits of P_b . We say a parent P_a of prefix P_b is an *identical parent* if P_a translates to the same route as P_b , i.e., packets matching both prefixes will be routed to the same next hop.

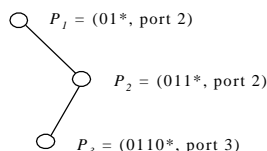


Figure 1: A pruning example.

The idea of pruning is fairly simple. If P_a is an identical parent of P_b , then P_b is a redundant routing prefix. To see this, assume the longest prefix matched for an IP address is P_b , then the IP address will match P_a as well by definition. If P_b is removed from the routing table, P_a becomes the longest matched prefix. Since they both translate to the same route, removing P_b makes no difference. Note that the technique is general enough that

it can be used in any routing lookup algorithm, regardless of how the routing table is stored.

An example of pruning is shown in Figure 1. The routing table is organized as a tree structure. Prefix P_1 is an identical parent of P_2 , and P_2 is a parent of P_3 . In this simple example, P_2 is a redundant prefix, thus can be removed without affecting routing functionality.

2.2. Mask extension

The second technique exploits the flexibility given by TCAM hardware. The mask for a routing prefix stored in TCAM consists of a number of 1s (same number as the prefix length), followed by all 0s. However, TCAM allows arbitrary mask to be used, i.e., they do not have to be continuous bits of 1s or 0s. We call this technique *mask extension* simply because it extends the mask to be any arbitrary combination of 1s and 0s.

We describe the mask extension technique using a simple example. P_1 and P_2 in Table 1 both correspond to the same route: port 7. It is possible to combine the two prefixes into one single entry with prefix set to 10011100 and mask set to 11101100. The 0 at bit 4 in the mask prevents comparison at that bit, and allows matching of P_1 and P_2 in the same entry. Using only the mask extension technique, the compacted version of the original routing table in Table 1 is shown in Table 3. The table size has been reduced from 7 entries to 5 entries.

Table 3: Compacted table using mask extension

#	Prefix	Mask	Next Hop port
$P_1 \& P_2$	10001100	11101100	7
$P_1 \& P_3$	10011100	10111100	7
P_4	10001000	11111000	5
$P_5 \& P_7$	10010000	10110000	4
P_6	11110000	11110000	7

We note that the mask extension technique can be simply reduced to a logic minimization problem. In the following, we use *cube* to refer to the combined single entry for several prefixes, and we use *cover* to refer to the set of cubes that cover all prefixes. The problem then becomes: “Given a set of prefixes with the same length and same route, find a minimal cover.”

It is well known that logic minimization problem is an NP-complete problem. Thus it leaves little hope to find an efficient exact algorithm. Fortunately, fast heuristic algorithm exists which produces a near optimal solution with finite computing resources. We choose to use ESPRESSO-II [17] which is proven to be a very efficient implementation.

We now show how the mask extension technique could be implemented. First, we define the following terms:

- $P(l, n)$: The set of prefixes that have length l and next hop port n

- $C(l, n)$: The set of cubes that cover $P(l, n)$. It is the result of logic minimization
- S_{on} : The “on-set” for logic minimization.
- S_{dc} : The “don’t care set” for logic minimization
- $F(P)$: The set of cubes that cover prefix P

The pseudo code to compact a routing table using mask extension is shown in Figure 2. The routine finds the set of routing prefixes with the same prefix length and route, then uses ESPRESSO-II minimization algorithm to compute the minimal cover. **minimize()** is the ESPRESSO-II logic minimization routine described in [17]. It takes two arguments: the on-set to be minimized and the don’t care set. **insertToCAM()** stores the compacted table into TCAM array for routing lookup.

```

Compact_routing_table ()
  For all prefix length  $l$ 
    For all possible next hop  $n$ 
       $C(l, n) = \text{minimize}(P(l, n), nil)$ ;
      insertIntoCAM(  $C(l, n)$  );

```

Figure 2: Pseudo code to compact routing table using mask extension

3. Incremental update

The routing table is hardly static. In fact, there could potentially be hundred of updates (insert/withdraw) per second[20]. Most routing updates are route flaps, i.e., the same prefix is added then removed repeatedly in quick succession. It is straightforward to reduce the number of actual updates by keeping a buffer of recent route update announcements, and only update the end result, thereby eliminates most route flaps. Still, tens of updates could be required per second in backbone routers, thus fast update algorithm is necessary. Incremental update for the pruning technique is quite straightforward, thus we omit further discussion. However, incremental update for mask extension is non-trivial. In the following, we describe fast update algorithm for mask extension technique.

3.1. Insertion

When new prefix arrives, we could re-minimize the whole set $P(l, n)$ along with the new prefix, although the computation will be very high. Another naïve approach will be inserting the new prefix directly into the TCAM array (since a prefix is itself a cube, although not necessarily the largest cube). The simple-minded algorithm will reduce the compaction ratio over time, results in almost no area savings at all.

A heuristic minimization algorithm is used in our implementation to facilitate fast incremental insertion. When a new prefix P is inserted, we compute a minimal cover C_p using P as the on-set of minimization, and using

the existing compacted table as don’t-care set. The minimal cover C_p will be inserted into the TCAM array instead of the new prefix P . Because existing entries in TCAM array could be redundant as a result of the insertion of C_p , we examine each existing entry in turn, and remove entries completely covered by C_p . The algorithm is shown in Figure 3.

```

Insert_prefix (P)
   $S_{on} = \{ P \}$ ;
   $S_{dc} = C(l, n)$ ;
   $C_p = \text{minimize}(S_{on}, S_{dc})$ ;
  For each  $C$  in  $C(l, n)$ 
    /* see if  $C$  is a subset of  $C_p$  */
    If ( subset( $C, C_p$ ) )
      /* remove  $C$  if duplicate */
      remove( $C, C(l, n)$  );
      /* also remove it from CAM array */
      removeFromCAM(  $C$  );
   $C(l, n) = C(l, n) + \{ C_p \}$ ;
  insertIntoCAM(  $\{ C_p \}$  );

```

Figure 3: Incremental insertion algorithm

As an example, if we are inserting a new entry $P=(110011--, 7)$ into the routing table shown in Table 1. $S_{on} = \{110011--\}$, $S_{dc} = \{100-11--, 1-0111--\}$, and after minimization $C_p=1-0-11--$. The subsequent redundancy check will remove 100-11-- and 1-0111-- from TCAM because they are completely covered by the new cube C_p . The resulting $C(6, 7)$ is $\{1-0-11--\}$.

3.2. Withdrawal

When a prefix is removed from the routing table, the algorithm will be more complex because the prefix could be covered by a number of cubes. It is necessary to remove all cubes covering the prefix, then recalculate a minimum cover from the affected prefixes. To see this, refer to the example shown in Figure 4. C_1, C_2, C_3 are cubes, and P_1, P_2, P_3, P_4 are prefixes. A line between them means that the cube covers the corresponding prefix.

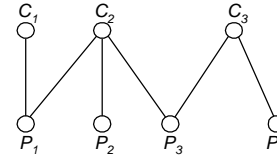


Figure 4: Example cube and prefix relationship

If P_3 needs to be removed, then C_2 and C_3 have to be removed, as a result, P_2 and P_4 do not have any cover anymore, thus it is necessary they are included in the computation for new cover. Note that P_1 is not affected because even though C_2 is removed, it is still covered by C_1 . Our incremental removal algorithm searches for such prefixes that are no longer covered by any cubes, then includes them in the computation for new cover. The algorithm is shown in Figure 5. As in the incremental

insertion algorithm, it is necessary to eliminate redundant cubes.

```

Withdraw_prefix (P)
/* search for prefix affected by removal of P */
For each C in F(P)
  remove(C, C(l, n));
  removeFromCAM(C); /* also remove from CAM */
  For each P covered by C
    remove(C, F(P∅));
    If (F(P∅) = nil)
      Son = Son + {P∅};
  Sdc = C(l, n);
  set = minimize(Son, Sdc)

/* remove redundant cubes */
For each C in C(l, n)
  /* if C is a subset of set, remove it */
  If (subset(C, set))
    remove(C, C(l, n));
    removeFromCAM(C);
  C(l, n) = C(l, n) + set;
  insertIntoCAM(set);

```

Figure 5: Incremental withdrawal algorithm

As an example, if we are removing P_l from Table 1, $S_{on} = \{100011--, 110111--\}$, $S_{dc} = nil$, and after minimization $set = \{100011--, 110111--\}$. The subsequent loop can not find any redundant cube, thus the resulting $C(6, 7)$ is $\{100011--, 110111--\}$.

4. Experimental result

We implemented the algorithms described in this paper, and evaluated its performance on a number of routing tables found in backbone routers located at major Internet eXchange Points (IXP). The routing table snapshot was captured by IPMA project [8] on Mar. 7, 2001.

We first used pruning to remove redundant routing prefixes, then we use mask extension technique to further reduce the table size. The ESPRESSO-II algorithm we used in the mask extension step has several options to control the minimization quality and its runtime. The *exact* option enables finding the minimal number of cubes, although not necessarily the minimal number of literals. This is the optimum solution for our problem because each cube corresponds to one TCAM entry, minimizing number of cubes is equivalent to minimizing the whole routing table size. Reducing the number of literals does not reduce routing table size further. Theoretically, the runtime could be quite high for exact minimization, in fact, if the pruning technique is not used, the runtime for mask extension step takes twice as long when exact minimization is enabled. But, when pruning technique is used, the routing table size is substantially smaller, and different minimization options showed no noticeable difference in terms of running time. So, we

only show results with exact minimization enabled as it produces slightly better compaction result at no extra cost.

The original table size, the size after pruning, and the size after pruning and mask extension are shown in Table 4, along with the compaction ratio and the computation time. The pruning technique alone can consistently reduce the routing table size by roughly 25%. When mask extension technique is further applied, the overall compaction ratio achieved range from 42.7% up to 48%. Mask extension roughly contributes an additional 20% saving. Although not shown, mask extension alone (without pruning) can reduce the table size from 27.3% up to 30.4%, but with a higher runtime because of the larger input for logic minimization. The ESPRESSO-II algorithm exhibits exponential runtime with respect to the input size, so applying pruning technique first helps to reduce the runtime of mask extension step. The overall runtime shown in the table is measured on a Pentium III 500Mhz PC platform. The runtime is quite large for large routing tables because of the exponential behavior of ESPRESSO-II.

Table 4: Routing table compaction result

IXP	Mae East	Mae West	pacbell	paix	aads
original size	23554	32139	38791	15906	29195
after pruning	17791	24741	28481	11828	21857
% saving	24.5%	23%	26.6%	25.6%	25.1%
pruning + mask extension	13492	18105	20166	8930	16057
% saving	42.7%	43.7%	48%	43.9%	45%
Run time (s)	14.2	49.3	77	8.26	44.1

It is possible to further increase the compaction ratio using prefix expansion [11]. Prefix expansion reduces the number of possible prefix length to a fixed small number, and at the same time, increases the size of on-set at each chosen prefix length, which results in better chance of being minimized further. However, the runtime required is substantially larger because of the huge input, thus limits its practical usage.

The time required to do the compaction could be quite high for large routing tables. The major portion of the runtime is spent on prefix length with the largest number of prefixes. For example, most of the computation time for Pacbell table is spent on prefixes with length 24. The largest number of inputs for minimization for a particular port reached 11634 entries. The runtime could be reduced by breaking the input into smaller sections, minimizing them individually, then combining the result together for final minimization.

Because of the long runtime of mask extension technique, it should be done only once at system initialization, then the incremental update algorithm should be used for routing updates. As mentioned before, incremental update for pruning technique is trivial, so we

do not evaluate its performance here. Instead, we use only mask extension to compact routing table, and then apply the incremental update algorithm in section III to evaluate its performance. We used a 12 hour routing update trace captured from MaeEast IXP on Dec. 1, 2000. The trace contains close to 35000 updates. The original routing table size and the compacted routing table size are plotted after each insertion and withdrawal. The result is shown in Figure 6. Note that they are shown on different scale. Clearly, the size of the compacted table closely follows the size of the original table, with the gap remaining fairly constant. It suggests that our incremental update algorithm is capable of maintaining the compaction ratio. In fact, at the end of the trace, the gap between the two lines increases by close to 300, possibly because the larger table at the end presents better optimization opportunity. In our implementation, the average run time for each update is 22 ms on a Pentium III 500 Mhz processor, enough to support up to 50 updates per second. Combined with route flaps buffering mechanism, it can handle updates seen in the Internet backbone.

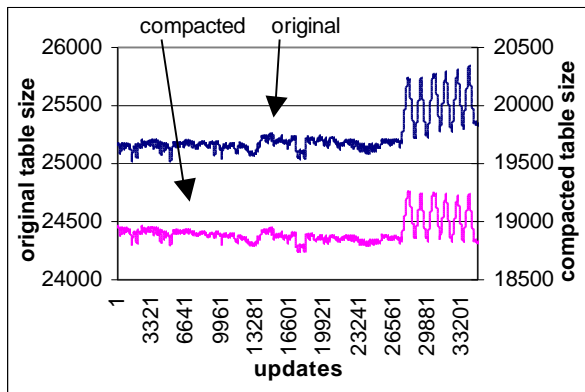


Figure 6: Table size after incremental updates

5. Conclusion

Because of the speed limitation of software based lookup algorithm, Ternary CAM (TCAM) is being used more often to solve longest prefix matching lookup problem in the state-of-art high speed router design. In this paper, we presented two techniques to compact a routing table stored in TCAM. Our simulation result shows up to 48% space saving, which directly translates to cost saving because either less TCAM chips are required or a smaller capacity TCAM could be used. In addition, our approach enables less power consumption and less heat dissipation. The techniques can also help routers to effectively cope with routing prefixes explosion. Fast incremental update algorithm has been presented which is capable of maintaining space saving even after a large number of updates.

References

- [1] Y. Rekhter, T. Li, "An Architecture for IP Address Allocation with CIDR," RFC 1518, 1993
- [2] S. Nilsson, G. Karlsson, "IP-address lookup using LC-tries," *IEEE Journal on Selected Areas in Communications*, vol. 17, no. 6, pp. 1083-1092, 1999.
- [3] P. Gupta, B. Prabhakar and S. Boyd, "Near-Optimal Routing Lookups with Bounded Worst Case Performance," *Proc. Infocom*, vol. 3, pp 1184-92, March 2000, Tel Aviv, Israel.
- [4] D. Shah and P. Gupta, "Fast Updates on Ternary-CAMs for Packet Lookups and Classification," *Proc. Hot Interconnects VIII*, August 2000, Stanford.
- [5] G. Bollano, S. Claretto, E. Filippi, A. Torielli and M. Turolla, "Merging Hardware and Software: Intellectual Property Cores for Internet Applications," *Proc. IEEE CICC*, 2000, Orlando, Florida.
- [6] P. Gupta, S. Lin and N. McKeown, "Routing Lookups in Hardware at Memory Access Speeds," *Proc. Infocom*, April 98, San Francisco.
- [7] T. Hayashi and T. Miyazaki, "High-Speed Table Lookup Engine for IPv6 Longest Prefix Match," *Proc. Globecom*, 1999, vol. 2, pp 1576-1581
- [8] Merit Networks, Inc., <http://www.merit.edu/ipma>
- [9] M. Waldvogel, G. Varghese, J. Turner, B. Plattner, "Scalable High-Speed IP Routing Lookups," *Proc. ACM SIGCOMM 1997*, pp. 25-36, Cannes, France.
- [10] A. Brodnik, S. Carlsson, M. Degermark, S. Pink, "Small Forwarding Tables for Fast Routing Lookups," *Proc. ACM SIGCOMM 1997*, pp. 3-14, Cannes, France.
- [11] V. Srinivasan and G. Varghese, "Fast Address Lookups Using Controlled Prefix Expansion," *ACM Transactions on Computer Systems*, Vol. 17, no. 1, pp. 1-40, Oct. 1999
- [12] Netlogic microsystems, "Ternary Synchronous Content Addressable Memory (IPCAM)," <http://www.netlogicmicro.com/pdf/NL82721.pdf>
- [13] S. Nilsson and G. Karlsson, "IP-address Lookup using LC-tries," *IEEE Journal on Selected Areas in Communications*, Vol. 17, no. 6, pp. 1083-92, 1999.
- [14] T.B. Pei and C. Zukowski, "VLSI Implementation of Routing Tables: Tries and CAMs," *Proc. Infocom 91*
- [15] A. McAuley and P. Francis, "Fast Routing Table Lookup Using CAMs", *Proc. Infocom 93*, Vol. 3, pp. 1382-91, Mar. 1993.
- [16] M. Kobayashi, T. Murase and A. Kuriyama, "A Longest Prefix Match Search Engine for Multi-Gigabit IP Processing", *Proc. ICC 2000*, June 2000.
- [17] R.K. Brayton, et al., *Logic Minimization Algorithms for VLSI Synthesis*, Kluwer Academic Publishers, Boston, MA., 1984
- [18] I. Hsiao and C.W. Jen, "A New Hardware Design and FPGA Implementation for Internet Routing towards IP over WDM and Terabit Routers", *Proc. IEEE Symposium on Circuits and Systems*, Vol. 1, pp. 387-390, 2000
- [19] T.C. Chiueh and P. Pradhan, "Cache Memory Design for Network Processors," *Proc. High Performance Computer Architecture*, pp. 409-418, 1999.
- [20] C. Labovitz, G.R. Malan and F. Jahanian, "Internet Routing Instability," *The IEEE/ACM Transactions on Networking*, Vol. 6, no. 5, pp. 515-528, 1999.