Reducing TCAM Power Consumption and Increasing Throughput

Rina Panigrahy, Samar Sharma
Cisco Systems
{rinap, ssharma}@cisco.com

Abstract

TCAMs have been an emerging technology for packet forwarding in the networking industry. They are fast and easy to use. However, due to their inherent parallel structure they consume high power - much higher than SRAMs or DRAMs. A system using four TCAMs could consume upto 60 watts. The power issue is one of the chief disadvantages of TCAMs over RAM based methods for forwarding. For a system using multiple TCAMs we present methods to significantly reduce TCAM power consumption for forwarding, making it comparable to RAM based forwarding solutions. Using our techniques one can use a TCAM for forwarding at 3 to 4 watts worst case.

Our techniques also have an interesting connotation to TCAM forwarding rates. For a static distribution of requests we present methods that make the forwarding rate of a system proportional to the number of TCAMs. So if a system has four TCAMs, one could achieve a four fold performance of that of a single TCAM for a static distribution of requests.

1. Introduction

Currently, TCAM based solution is much faster than DRAM based solutions for packet forwarding and classification. For example, for IPv6 forwarding, DRAM based solution can perform 20 Million packets per second (MPPS) [7], whereas TCAM based solution can perform 120 MPPS[14][20]. Similarly TCAMs can achieve much higher number of lookups per second for packet classification.

In order to support large number of layer 3 prefixes (IP, IPX), 4 to 8 TCAM chips are often used. In current implementations, a longest prefix match involves issuing lookups to each of the TCAM chips. Since each of the components is being used for every lookup, the total power consumed by the TCAM subsystem is proportional to the number of TCAM chips used. For longest prefix match, we provide techniques, where for each lookup only one chip needs to be searched and the others can be inactive. This means that every longest prefix match triggers a lookup in only one of the chips. We assume for illustration that there are 8 chips. Since this translates into performing only 1/8th of the lookups per chip on an average, it could result in significant reduction in power consumption.

Alternatively, by placing each TCAM on a separate bus, the scheme can also be used to provide higher number of lookups/s than that supported by a single TCAM chip. If the distribution of traffic among IP addresses is known, than one can partition the prefixes across the 8 TCAMs so that each TCAM handles close to 1/8th of the traffic. This can be used to deliver an aggregate throughput close to 8*125 M lookups/s, i.e., 1000 M lookups/s.

We also introduce the concept of paged TCAM to achieve significantly lower power consumption within each TCAM chip.

2. Algorithm Overview

The basic idea is to partition the set of prefixes into 8 groups so that for a prefix lookup one can a priori easily decide that the prefix will match in only one of the groups. We are thus pruning the search into one of the groups.

2.1 Pruned Search

Let's say we are using the TCAMs for IPv4 lookups. Take the 32 bit IP address. Use the first 3 bits (msb) of the IP address as a group id. Only turn on (issue lookup to) the chip whose group id equals these first 3 bits. Assuming that the IP prefixes are divided about equally into the 8 different groups, about 1/8th of the prefixes reside per chip (group) and the total number of lookups per second is reduced by a factor of 8. However this assumption may not be true in practice; for example in an enterprise network, most IP prefixes might begin with the same first 3 bits (msb). So another heuristic might be to use some other three bits (say, least significant bits of the most significant byte).

We present a scheme to partition the prefixes into groups of about equal size. An IP address could lie in the space 0 to (2^{32}-1). We partition this space into 8 disjoint ranges so that each range maps to about 1/8th of the total number of prefixes. We say a prefix falls into a range if any IP addresses from that range can match the prefix. Formally, a prefix P* falls in a range [a, b] if the range [P0..0, P1..1] intersects with the range [a, b]. Some prefixes can fall into multiple ranges. For example, the default prefix **** falls in all ranges. However the number of such prefixes can be shown to be small (at most 32*7 = 224). One way to check if a prefix P* falls only in a single range [a, b] is to check if the following inequalities hold.

$$a < P0..0,$$
$$P1..1 < b$$

This will ensure that P* can not fall into any other range, because the ranges are disjoint.

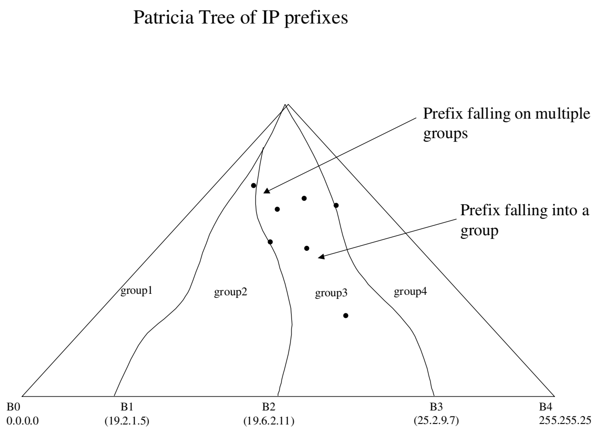
Patricia Tree of IP prefixes
Prefix falling on multiple groups
Prefix falling into a group
group1
group2
group3
group4
B0
0.0.0.0
B1
(19.2.1.5)
B2
(19.6.2.11)
B3
(25.2.9.7)
B4
255.255.25

Figure 1. Partitioning prefixes into four equal size groups

Let B_1, B_2,... denote the boundary points that partition the IP address number line. We choose these boundaries in such a way about an equal number of prefixes fall into each range (Figure 1). We place each group of prefixes in a separate TCAM chip. As for prefixes that fall into multiple ranges, we put them into the chips that correspond to those ranges. It is easy to show that there are at most 32*2 = 64 boundary prefixes that need to be present in a TCAM chip (these prefixes fall into multiple ranges). This is because any prefix that falls into multiple ranges must be on the path from one of these boundaries to the root. Otherwise, it will strictly lie in the interior of one of the regions carved out by these paths. More formally if a prefix P is not matched by any of B_1, B_2,... then there is some i so that P is "strictly to the right of" the path from B_i to root and "strictly to the left of" path from B_{i+1} to the root.

Clearly, total number of prefixes on these paths is at most 7*32. Each range will need to include, besides prefixes in the interior, the prefixes on the two boundary paths. The total number of these boundary prefixes for a given range is at most 2*32. So every TCAM will have at most 64 prefixes that fall into multiple ranges.

One will now need a set of 8 comparator pairs in the TCAM interface chip to determine the group from the input IP address (Figure 2). Only the TCAM chip corresponding to this group will be searched. Others will remain inactive.

One also needs to ensure that during updates, no group becomes too big. This might require moving prefixes across groups, which is addressed in the following subsection.

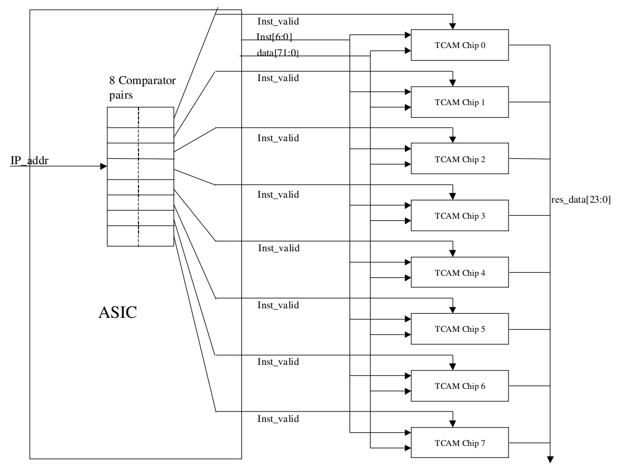

Figure 2. Pruned Search

2.1.1 Example of prefix partitioning

We now present an example of our methods for a system using a TCAM chip available today.

- Say each TCAM can perform 100 million lookups per second.
- There are 2M IPv4 addresses that need to be supported.
- Each address takes up 72 bit entry.
- Say, each TCAM has 256k entries of 72 bits each and consumes 14.3W in the worst case.
- That is, 8 chips, each of size 256k*72 bit are being used.
- So, the 8 chips, if all used for lookup (existing approach), would consume about 14.3W * 8 = 114W power.
- Using our scheme, the power consumption would be 14.3W for effectively one TCAM chip, and an idle power of 2.5W for the remaining 7 chips. So total worst case power consumption would be (14.3 W + 7*2.5W) = 31.8W.

2.2 Prefix Updates

During updates, one needs to maintain about an equal distribution of prefixes into the 8 groups. During an insert, we first find which group the prefix falls in. If it falls in multiple groups (rare) put it in all chips; otherwise put it in the free entry in the appropriate chip. If there is no free entry (rare), the inserts can be handled by borrowing a free entry from the next TCAM (Figure 3). This will involve shifting the boundary between the two groups. The new boundary can be easily computed by maintaining the prefix count with every node in the Patricia tree. This count keeps the number of prefixes in the sub-tree rooted at that node. If the next TCAM is also full, this could be done with at most 8 moves propagating across all TCAMs. Additionally, one also needs to maintain the prefix length ordering within a TCAM; that is, the longer prefixes need to be placed above the shorter ones as the TCAM returns the topmost matching prefix. Clearly, the insert within a TCAM chip is easier if there is no prefix ordering constraint (for example, in a priority based TCAM). It is also possible to design cleverer algorithms for updates to minimize duplication and moves, as described in next section.

As for an initial set of prefixes, one can partition this initial set by starting with an empty set and performing inserts.

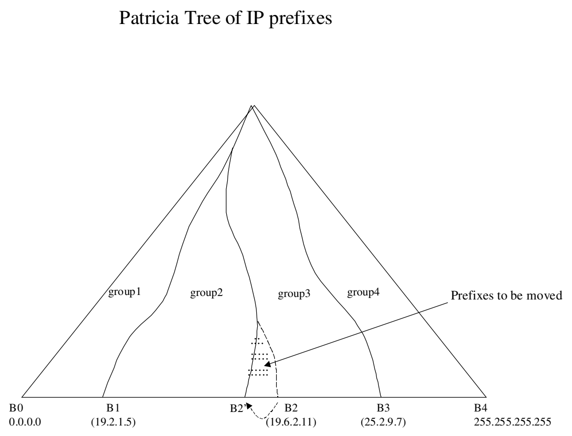

Figure 3. Moving prefixes during inserts if the TCAM is full

2.2.1 Update Algorithms

Now we describe two algorithms for performing prefix inserts.

First Algorithm

1. There are 8 boundary paths (in the tree)
2. Each TCAM always reserves 64 entries for its two boundary paths
3. If x is the actual number of prefixes on these paths (max can be 64) in a TCAM chip, then (64 - x) virtual entries are reserved in the TCAM.
4. If an insert is on these boundary paths, then use one of the virtual entries in the TCAM(s).
5. If it is not on the boundary path, and there is a free entry, or there are prefixes outside the boundary path (because deletes are lazy), use it.
6. If there is no free entry, then shift boundary as follows :
7. Look at any neighboring boundary B, say to the right. Let P be the first prefix strictly to the left of boundary path from B to root (ordering of prefixes is defined to be the "post-order" traversal of the Patricia tree). Set new boundary B' to be equal to P1….1. Now we have reduced the number of internal prefixes in the TCAM by at least one.
8. Insert all prefixes on B' to root into adjacent TCAM.

Worst case time for one insert : 32 moves per boundary
Worst case time for x successive inserts to same TCAM : (32+x) moves per boundary. As x grows large, amortized cost is 1.

Second Algorithm (requires one move per insert per boundary) :

1. Keep diffused boundary paths. That is, there are two nodes N_1 and N_2 on the tree and the boundary consists of paths between N_1 to root and N_2 to root and all the nodes between these paths. There is at least one node between N_1 and N_2 that is a 32-bit number. It is this 32-bit number that is programmed as the boundary B in the comparator. TCAMs on either side of the boundary are required to keep prefixes on the boundary.
2. Number of prefixes on a boundary is at most 64. A TCAM will keep 128 entries reserved, some being virtual, for the boundary prefixes.
3. If an insert is on a boundary, then use a virtual entry.
4. Otherwise, if there are free entries or there are entries outside the boundaries, use them.
5. Otherwise, look at any one boundary (N_1, N_2), say to the right.
6. Locate the first prefix P strictly to the left of the boundary.
7. Set N_1 = P and N_2 = first prefix to the left of previous value of N_2.
8. Insert P into the next TCAM.
9. Now we have reduced the number of internal prefixes in the TCAM by at least one.

10. Set B' equal to a 32-bit number between N_1 and N_2. Note that B' is the quantity that stored in the comparator.

3. Improving the Lookup Performance

The technique can also attempt to provide a higher number of lookups per second than that supported by a single TCAM chip, assuming that the searches are about evenly distributed among TCAMs. Even if the searches are not evenly distributed across the 8 ranges, one can further subdivide each of these ranges and "randomly" distribute these sub-ranges across the TCAMs so as to ensure about an even distribution of prefixes and about an even distribution of lookups (traffic).

In particular, for a given distribution of traffic to destination IP addresses, it is possible to partition the entire space into about 4*8 ranges, and distribute these ranges into 8 TCAMs, so that each TCAM gets close to 1/8th of the number of prefixes and close to 1/8th of the traffic. This means for that given distribution, the 8 TCAMs can deliver close to 8*125MPPS = 1000MPPS.

Note that this is achieved by placing each TCAM on separate bus.

3.1 Algorithms for higher throughput

Assumptions :

1. No IP address gets more than 1/16 of total bandwidth (1000MPPS)
2. For each IP address, we know the bandwidth that it is using.

Algorithm 1 :

1. First obtain 4*8 ranges so that each range has at most 1/16th of the prefixes and 1/8th traffic. This is achieved by following steps :
a) First partition the IP addresses in 2*8 ranges based on traffic, where each range has at most 1/8th traffic.
b) Then take these 2*8 ranges and subdivide them such that a range of size x gets divided into (x/(1M))*2*8 ranges, assuming there are a total of 1 Million prefixes.
c) Total number of ranges = 4*8.
2. Divide these 32 ranges between TCAMs so that each TCAM does not get too many prefixes and too much traffic. Randomly pick 4 ranges to be inserted in first TCAM, another four ranges in second TCAM and so on.
3. Put these ranges in 32 comparators.

Analysis

Since the distribution is random, one can expect with reasonable probability that about 1/8th of the traffic goes to each TCAM.

The following analysis shows that the algorithm can handle 250MPPS in the **worst case**.

1. Now each TCAM can get at most 1M*((32/8)*(1/16)) = (1M)/4 prefixes. => we need 2M worth of TCAM to support 1M prefix entries.
2. We can support half of total TCAM bandwidth (i.e. (1000/4)MPPS = 250MPPS)
3. By doubling the number of entries in each TCAM and speed of each TCAM, we can support 500MPPS.
4. Alternatively, using the current TCAM, we can support half the number of entries at half of 500MPPS rate.

Algorithm 2 :

1. Same as step 1 in Algorithm 1.
2. Just sort the ranges by C_i/P_i. Greedily fill up a bin to minimize error.
3. Put these ranges in the array of 32 comparator pairs.

Using 4*8 comparator pairs, we can guarantee that no more than (1/8)*(1+1/4) traffic. Each TCAM gets (1/8)*(1+1/8) prefixes.
This can be improved by using more comparator pairs.

Advantages :

1. The scheme provides much higher throughput
2. The scheme works well for IPv6 prefixes as well

4. Paged TCAM

We now apply the ideas described above inside a single TCAM chip to achieve significantly lower power consumption by each TCAM chip. We show successive modifications to the TCAM hardware to achieve this.

1. Assume that the TCAM has 256k entries. We organize a TCAM into pages where each page contains about 256 consecutive entries. This will give rise to 1024 pages. For IP prefix lookup, supply 6 bits in addition to the 32bit key. With each page in the TCAM, associate a 6bit group ID. During a lookup, turn on a page only if its group id matches the first 6 bits of the input key. By generating the 6bits appropriately, one can ensure that roughly 1/64th pages in the TCAM are active. The 6 bit id is generated by dividing the range of prefixes into 64 roughly equal sized chunks. This can be accomplished by using a set of comparator pairs

external to the TCAM containing 64 ranges. Updates can be handled as described earlier. We might also want to provide a null (could be thought of as a masked group match) group-id with a page so that it is always active. This would be useful for prefixes that match multiple groups (eg, 1******). That is, a group-id could have "don't-care" bits.

2. Put the array of comparator inside the paged TCAM. See Figure 4.
3. Put multiple sets of comparator arrays (say 2). Each comparator array takes a different portion of the key and each array produces 6 bits. Each page can decide which arrays 6 bits it will use for comparing with its group id. This could be useful for organizing ACL (Access Control list) or QoS (Quality of Service) entries into pages. One of the comparator arrays would classify based on the source IP address and the other based on the destination address. Each page could be configured to choose the group id produced by either of the comparator arrays.

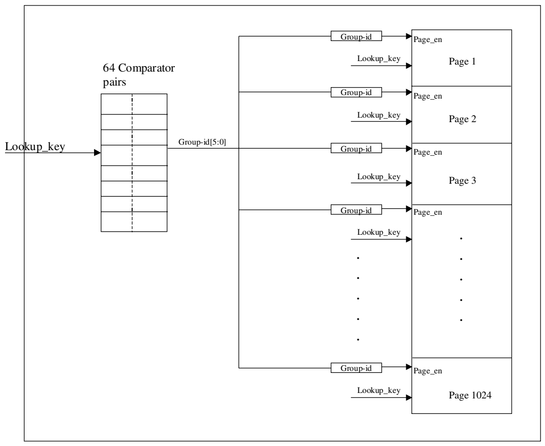

Figure 4. Paged TCAM

5. Conclusions

In order to support large number (say, 1 million) of layer 3 prefixes, 4 to 8 TCAM chips are often used. In current implementations, a longest prefix match involves issuing lookups to each of the TCAM chips. Since each of the components is being used for every lookup, the total power consumed by the TCAM subsystem is proportional to the number of TCAM chips used. For longest prefix match, we provide techniques, where for each lookup only on chip needs to be searched and the others can be inactive.

Alternatively, by placing each TCAM on separate bus, the scheme can also be used to provide higher number of lookups/s than that supported by a single TCAM chip. If the distribution of traffic among IP addresses is known, then one can partition the prefixes across the 8 TCAMs so that each TCAM handles close to 1/8th of the traffic. This can be used to deliver an aggregate throughput close to 8*125 M lookups/s, i.e., 1000 M lookups/s.

We also introduce the concept of paged TCAM to achieve significantly lower power consumption within each TCAM chip.

7. References

[1] A. Brodnik, S. Carlsson, M. Degermark, and S. Pink, “Small forwarding tables for fast routing lookups,” in *Proceedings of ACM SIGCOMM*, Oct. 1997, pp. 3-13.

[2] P. Gupta, S. Lin, and N. McKeown, “Routing lookups in hardware at memory access speeds,” in *Proceedings of INFOCOM*, March 1998, pp. 1240-1247.

[3] Pankaj Gupta, “Routing lookups and packet classification: theory and practice,” August 2000, Tutorial at *Hot Interconnects VIII*, Stanford.

[4] Donald E. Knuth, “The Art of Computer Programming : Sorting and Searching,” Addison-Wesley Pub Co.

[5] M. Kobayashi, T. Murase, and A. Kuriyama, “A longest prefix match search engine for multi-gigabit ip processing,” in *Proceedings of ICC 2000*, June 2000.

[6] C. Labovitz, G.R. Malan, and F. Jahanian, “Internet routing instability,” *The IEEE/ACM Transactions on Networking*, vol. 6, no. 5, pp. 515-528, Oct. 1999.

[7] Butler W. Lampson, V. Srinivasan, and George Varghese, “IP Lookups using Multiway and Multicolumn Search,” *IEEE/ACM Transactions on Networking*, vol. 7, no. 3, June 1999.

[8] University of Michigan and Merit Network, “Internet Performance Measurement and Analysis (IPMA) project,” www.merit.edu/ipma.

[9] D. R. Morrison, “PATRICIA - Practical Algorithm to Retrieve Information Coded in Alphanumeric,” *Journal of the ACM*, 15(4), pp514-534, Oct 1968.

[10] S. Nilsson and G. Karlsson, “IP-address lookup using LC-tries,” *IEEE Journal on Selected Areas in Communications*, vol. 17, no. 6, pp. 1083-1092, 1999.

[11] Wooguil Pak, Saewoong Bahk, “Flexible and fast IP lookup algorithm,” in *Proceedings of ICC 2001*, June 2001.

[12] Rina Panigrahy, and Samar Sharma, “Avoiding Patricia Tree and Region Management while using Ternary Content Addressable Memory,” *Cisco Document*.

[13] Rina Panigrahy, Samar Sharma, and Suran De Silva, “A System for Caching Forwarding Entries Selectively and Dynamically,” *Cisco Document*.

[14] Music Semiconductors, “Application Notes”, http://www.music-ic.com

[15] Devavrat Shah and Pankaj Gupta, “Fast Updates on Ternary-CAMs for Packet Lookups and Classification,” *Proc. Hot Interconnects VIII*, August 2000, Stanford. *IEEE Micro*, vol. 21, no. 1, January/February 2001.

[16] Samar Sharma, Rina Panigrahy, and Abhijit Patra, "Techniques for Efficient TCAM Management for Longest Prefix Match Problems," *Cisco Document*.

[17] Samar Sharma, Rina Panigrahy, and Abhijit Patra, "Techniques for Efficient Location of Free Entries for TCAM Inserts," *Cisco Document*.

[18] V. Srinivasan, S. Suri, and G. Varghese, "Packet classification using tuple space search," *ACM Computer Communication Review*, 1999. *ACM SIGCOMM'99*, Sept. 1999.

[19] 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.

[20] Sibercore Technologies, "Application Notes", http://www.sibercore.com/products_AppNotes.htm

[21] Telstra, http://www.telstra.net/ops/bgp/bgp-active.html

[22] M. Waldvogel, G. Varghese, J. Turner, and B. Plattner, "Scalable high-speed ip routing lookups," in *Proceedings of ACM SIGCOMM*, Oct. 1997, pp. 25-36.