TCAM Architecture for IP Lookup Using Prefix Properties

TERNARY CONTENT-ADDRESSABLE MEMORY (TCAM) IS A POPULAR HARDWARE DEVICE FOR FAST ROUTING LOOKUP. THOUGH USEFUL, TCAM ARRAYS HAVE HIGH POWER CONSUMPTION AND HEAT DISSIPATION, PROBLEMS ALLEVIATED BY REDUCING THE NUMBER OF ROUTING-TABLE ENTRIES. THE AUTHORS PRESENT AN APPROACH THAT EXPLOITS TWO PROPERTIES OF PREFIXES TO COMPACT THE ROUTING TABLE.

Ravikumar V.C.
Rabi N. Mahapatra
Texas A&M University

In modern IP routers, Internet Protocol (IP) lookup forms a bottleneck in packet forwarding because the lookup speed cannot catch up with the increase in link bandwidth. Ternary content-addressable memories (TCAMs) have emerged as viable devices for designing high-throughput forwarding engines on routers. Called ternary because they store don't-care states in addition to 0s and 1s, TCAMs search the data (IP address) in a single clock cycle. Because of this property, TCAMs are particularly attractive for packet forwarding and classifications. Despite these advantages, large TCAM arrays have high power consumption and lack scalable design schemes, which limit their use.

Today's high-density TCAMs consume 12 to 15 W per chip when the entire memory is enabled. To support the superlinearly increasing number of IP prefixes in core routers, vendors use up to eight TCAM chips. Filtering and packet classification would also require additional chips. The high power consumption of using many chips increases cooling costs and also limits the router design to fewer ports.[1]

Recently, researchers have proposed a few approaches to reducing power consumption in TCAMs,[1,2] including routing-table compaction.[3,4] Liu presents a novel technique to eliminate redundancies in the routing table.[3] However, this technique takes excessive time for update because it is based on the Espresso-II minimization algorithm,[5] which exponentially increases in complexity with the number of prefixes in a routing table. Thus, our work's main objective is a TCAM-based router architecture that consumes less power and is suitable for the incremental updating that modern IP routers need. Additionally, the approach we will describe minimizes the memory size required for storing the prefixes.

We propose a two-level pipelined architecture that reduces power consumption through memory compaction and the selective enablement of only a portion of the TCAM array. We also introduce the idea of prefix aggregation and prefix expansion to reduce the number of routing-table entries in TCAMs for IP lookup.

Published by the IEEE Computer Society

Today, TCAM vendors provide mechanisms to enable a chunk of TCAM much smaller than the entire TCAM array. Exploiting this technology reduces power consumption during lookup in the TCAM-based router. We also discuss an efficient incremental update scheme for the routing of prefixes and provide empirical equations for estimating memory requirements and proportional power consumption for the proposed architecture. Finally, we use traces from `bbnplanet`, `attcanada`, `utah`, and `is` routers to validate the proposed TCAM architecture's effectiveness.

Today, routing tables have approximately 125,000 entries and will contain about 500,000 entries by 2005, making optimal storage important.

Background and related work

We can categorize approaches to performing IP lookup as either software or hardware based. Researchers have proposed various software solutions for routing lookup.[6,7] However, these software approaches are too slow for use in complex packet processing at every router, mainly because they require too many memory accesses: Independent of their design or implementation techniques, software approaches take at least four to six memory accesses for a single lookup operation. Today's packet processing requires speeds of about 40 Gbps; software approaches achieve no more than 10 Gbps.[8]

Hardware approaches typically use dedicated hardware for routing lookup.[9,10] More popular techniques use commercially available content-addressable memory (CAM). CAM storage architectures have gained in popularity because their search time is $O(1)$—that is, it is bounded by a single memory access. Binary CAMs allow only fixed-length comparisons and are therefore unsuitable for longest-prefix matching. The TCAM solves the longest-prefix problem and is by far is the fastest hardware device for routing. In contrast to TCAMs, ASICs that use tries—digital trees for storing strings (in this case, the prefixes)—require four to six memory accesses for a single route lookup and thus have higher latencies.[1] Also, TCAM-based routing-table updates have been faster than their trie-based counterparts.

The number of routing-table entries is increasing superlinearly.[8] Today, routing tables have approximately 125,000 entries and will contain about 500,000 entries by 2005, so the need for optimal storage is also very important. Yet CAM vendors claim to handle a maximum of only 8,000 to 128,000 prefixes, taking allocators and deallocators into account.[11] The gap between the projected numbers of routing-table entries and the low capacity of commercial products has given rise to work on optimizing the TCAM storage space by using the properties of lookup tables.[3,4]

However, even though TCAMs can store large numbers of prefixes, they consume large amounts of power, which limits their usefulness. Recently, Panigrahy and Sharma introduced a paged-TCAM architecture to reduce power consumption in TCAM routers.[2] Their scheme partitions prefixes into eight groups of equal size; each group resides on a separate TCAM chip. A lookup operation can then select and enable only one of the eight chips to find a match for an incoming IP address. In addition, the approach introduces a paging scheme to enable only a set of pages within a TCAM. However, this approach achieves only marginal power savings at the cost of additional memory and lookup delay.

Other work describes two architectures, bit selection and trie-based, which use a paging scheme as the basis for a power-efficient TCAM.[1] The bit selection scheme extracts the 16 most significant bits of the IP address and uses a hash function to enable the lookup of a page in the TCAM chip. The approach assumes the prefix length to be from 16 to 24 bits. Prefixes outside this range receive special handling; the lookup searches for them separately. However, the number of such prefixes in today's routers is very large (65,068 for the `bbnplanet` router), and so this approach will result in significant power consumption.

In addition, the partitioning scheme creates a trie structure for the routing table prefixes and then traverses the trie to create partitions by grouping prefixes having the same subprefix. The subprefixes go into an index TCAM, which further indexes into static RAM to identify and enable the page in the data TCAM that stores the prefixes. The index TCAM is quite large for smaller page sizes and is a key factor in power consumption. The three-level architecture, though pipelined, introduces considerable delay.

It is important to note that both the approaches[1,2] store the entire routing table, which is unnecessary overhead in terms of memory and power. Although the existing approaches reduce power either by compacting the routing table or selecting a portion of the TCAM, our approach reduces power by combining the two approaches and exploiting the observable properties of prefixes.

Prefix properties for compaction

The two purposes for delving into the prefix properties are

- to further compact the routing table beyond the compaction provided by existing techniques, and
- to derive an upper bound on minimization time to prevent it from becoming a bottleneck in the routing lookup.

Previous attempts to reduce the routing-table size using prefix properties used prefix overlapping and minimization techniques.[3] *Prefix overlapping* achieves compaction by storing only one of many overlapping prefixes that have the same next hop. Two prefixes overlap if one is an enclosure of the other. Let P_i be a prefix, with $|P_i|$ denoting the length of prefix P_i. Then $P_i \in P$ is called an enclosure of P_j, if $P_j \in P$, $j \neq i$ and $|P_i| < |P_j|$, such that P_i is a subprefix of P_j. If P_i and P_j have the same next hop, then P_i can represent them. Thus, set of overlapping prefixes $\{P_1, P_2, P_3, \ldots P_n\}$, such that any $|P_1| < |P_2| < |P_3| \ldots < |P_n|$ having the same next hop is replaceable with single entry P_1. If an update deletes entry P_1, entry P_2 should represent set of overlapping prefixes $\{P_2, \ldots P_n\}$. When an update adds new entry P_i such that $|P_i| < |P_1| < |P_2| \ldots < |P_n|$, then P_i replaces existing entry P_1. However, if new entry P_i arrives such that $|P_i| > |P_1|$, then the routing table should not change, except for an update of the set of overlapping prefixes.

Prefix minimization logically reduces two or more prefixes to a minimal set, if these prefixes have the same next hop. The logic minimization is an NP-complete problem, solvable using the Espresso-II algorithm. Espresso-II can minimize prefixes $\{P_1, P_2, P_3, \ldots P_n\}$ to $\{P'_1, P'_2, P'_3, \ldots P'_m\}$, such that $m \leq n$.

Although the prefix overlapping and minimization techniques together compact about 30 to 45 percent of the routing table, these techniques have an overhead when it comes to prefixes that require fast updates. The time taken for prefix overlapping is bounded and independent of the router's size. However, the logic minimization algorithm using Espresso-II has a runtime that increases exponentially with input size. Based on Liu's techniques,[3] the Espresso-II input can be as high as 15,146 prefixes for the **attcanada** router. Thus, the runtime for such a large amount of input data can be several minutes and is very expensive for incremental updates. We propose tackling this problem by establishing an upper bound, independent of router size, on the input to the minimization algorithm. We introduce another property, called prefix aggregation, to help us fix the upper bound. Based on this property, prefix minimization is time bounded.

Prefix aggregation

Figure 1 presents a portion of the routing-table traces from the **bbnplanet** router. It represents all the prefixes in the routing table, starting with 129.66 and having prefix length l, for $16 < l \leq 24$. The number 129.66 is the largest common subprefix (LCS) for the set of those prefixes. The LCS for any prefix is the subprefix with a length nearest to the multiple of eight, such that $|S_i| < |P_i|$, where S_i is the LCS of prefix P_i or LCS(P_i). The prefixes having a different LCS are least likely to have the same next hop. According to this observation, applying minimization to a set of prefixes having the same LCS should achieve a compaction nearly equal to that achieved by applying minimization to the entire lookup table.

So without sacrificing much compaction, the use of prefix aggregation can provide an upper bound on the number of possible pre-

Prefix	Next hop
129.66.6.0/24	4.0.6.142
129.66.8.0/24	4.0.6.142
129.66.12.0/24	4.0.6.142
129.66.20.0/24	4.0.6.142
129.66.21.0/24	4.0.6.142
129.66.30.0/23	4.0.6.142
129.66.31.0/24	4.0.6.142
129.66.32.0/19	4.0.6.142
129.66.34.0/24	4.0.6.142
129.66.47.0/24	4.0.6.142
129.66.48.0/24	4.0.6.142
129.66.64.0/18	4.0.6.142
129.66.88.0/24	4.0.6.142
129.66.95.0/24	4.0.6.142
129.66.111.0/24	4.0.6.142
129.66.128.0/22	4.0.6.142
129.66.132.0/24	4.0.6.142
129.66.172.0/24	4.0.6.142

Figure 1. Sample trace of routing table from **bbnplanet.**

Prefix	Next hop
10000001010000100000110XXXXXXXX	4.0.6.142
10000001010000100000100XXXXXXXX	4.0.6.142
10000001010000100000100XXXXXXXX	4.0.6.142
10000001010000100000100XXXXXXXX	4.0.6.142
10000001010000100000101XXXXXXXX	4.0.6.142
10000001010000100001111XXXXXXXX	4.0.6.142
10000001010000100001XXXXXXXXXXXX	4.0.6.142
10000001010000100001XXXXXXXXXXXX	4.0.6.142
10000001010000101000000XXXXXXXX	4.0.6.142
10000001010000101000000XXXXXXXX	4.0.6.142
10000001010000101010100XXXXXXXX	4.0.6.142

Figure 2. Result of applying prefix overlapping to Figure 1 trace.

fixes for input to the minimization algorithm. This input set is the largest set of prefixes such that each prefix in the set has the same LCS. These prefixes, which have LCS value S, can have prefix lengths from $|S|$ to $|S| + 8$. This range contains a maximum of 512 prefixes; however, all these combinations are not possible. For example, the assignment 120.255.0.0/12 precludes the combinations 120.255.0.0/13 through 120.255.0.0/16. So to calculate the maximum possible number of prefixes having the same LCS, we assign all combinations of 120.x.0.0/16 prefixes so that no assignments are possible for prefixes of length less than 16. Thus, a maximum of 256 prefixes can share the same LCS.

Prefix expansion

Researchers have presented the property of prefix expansion for IP lookup using software approaches.[12] We adopt this property to further compact the routing table and can represent the prefix expansion property as follows. If P_i represents a prefix, such that $|P_i|$ is not a multiple of eight, then the prefix expansion property expands P_i to $Pi \cdot Xm$, where X is a don't-care and $m = 8 - (|P_i| \bmod 8)$. The operator "$\cdot$" represents the concatenation operation. For example, the prefix 100000010100 expands to 100000010100XXXX using the prefix expansion property.

The Espresso-II algorithm will provide more compaction if all P_i are of same length. However, all the prefixes corresponding to the same LCS are not necessarily the same length. Hence, to increase the compaction, we expand prefixes to a size that is the nearest multiple of eight.

Just as the number of inputs governs Espresso-II's runtime, so does the input's bit lengths. We have already limited the input set to the set of prefixes with the same LCS. We next observe that, in terms of Espresso-II's calculations, the LCS is redundant. So it would be useful to further compact the prefixes by eliminating the LCS from each prefix and using the remainder, which is the least-significant 8 bits after prefix expansion.

Routing-table compaction

During router initialization, our approach compacts the routing table using the property of prefix overlapping and also removes redundant entries (prefix minimization). We then use the property of prefix aggregation to form sets of prefixes based on their LCS values. Each of these sets goes through prefix expansion before serving as input to Espresso-II.

Prefix overlapping. Prefix overlapping applies to all the prefix entries in the routing table with the same next hop. The routing table then stores only one of the many overlapping prefixes. Figure 2 is the result of applying prefix overlapping to the trace in Figure 1. In this case, prefix overlapping reduces the table by seven entries.

Prefix minimization and prefix expansion. Figure 3 shows that prefix minimization and prefix expansion reduce the number of

Prefix entries	Next hop
10000001010000010X0101100XXXXXXXX	4.0.6.142
1000000101000010OXX1010XXXXXXXXX	4.0.6.142
1000000101000010OXX01X00XXXXXXXX	4.0.6.142
10000001010000101000OX00XXXXXXXX	4.0.6.142
1000000101000010OXX00110XXXXXXXX	4.0.6.142
1000000101000010OXX1111XXXXXXXXX	4.0.6.142
100000010100001010000OXXXXXXXXXX	4.0.6.142
1000000101000010OX1XXXXXXXXXXXXX	4.0.6.142
1000000101000010O1001XXXXXXXXXXXXX	4.0.6.142

Figure 3. Result of applying prefix minimization and prefix expansion to Figure 1 trace.

Prefix entries	Next hop
10000001010000101010110OXXXXXXXX	4.0.6.142
1000000101000010000111111XXXXXXXX	4.0.6.142
10000001010000010000O1X00XXXXXXXX	4.0.6.142
1000000101000010000O1010XXXXXXXXX	4.0.6.142
1000000101000010100001OOXXXXXXXX	4.0.6.142
1000000101000010000001110XXXXXXXX	4.0.6.142
1000000101000010000O1111XXXXXXXXX	4.0.6.142
100000010100001010000OXXXXXXXXXX	4.0.6.142
1000000101000010001XXXXXXXXXXXXX	4.0.6.142
1000000101000010O1001XXXXXXXXXXXXX	4.0.6.142

Figure 4. Result of applying prefix minimization without prefix expansion to Figure 1 trace.

prefixes by nine. Figure 4 shows that prefix minimization without prefix expansion reduces the number of prefixes by eight.

Proposed architecture

The prefix properties reduce the IP lookup table's length (vertically) as described earlier. Here, we propose an architectural technique that reduces the IP lookup table laterally. This technique adopts the two-level routing lookup architecture in Figure 5.

Level 1 consists of a decoder and a reconfigurable page-enable block (PEB). The PEB is a combinational circuit consisting of a driver network to drive the output port, which connects directly to TCAM pages through page-enable lines. The size of the PEB's output port (in bits) equals the number of pages in the TCAM. The PEB is also reconfigurable to accommodate changes during a page overflow. Level 2 is a 256-segment TCAM array. In addition to the assigned pages, the TCAM array contains many empty pages; an implementation will use these pages for memory management.

For each incoming IP address, the first octet (A) goes to the decoder-PEB, which enables only one segment in the TCAM array. The decoder-PEB then sends page-enable signals to each TCAM page. The TCAM then compares the enabled pages with the next three incoming octets (B, C, and D) of the IP address.

The overhead of the decoder-PEB hardware is insignificant compared to that of the TCAM array. Avoiding the storage of the first octet in the TCAM array also reduces the total TCAM storage space by 25 percent. This savings is significant, especially when routing tables are as large as 125,000 entries. Also, the architecture enables only one segment of the TCAM array at any time, significantly reducing power consumption.

The lookup latency is the TCAM read access time plus small gate delays because of the decoder-PEB hardware. But gate delays should be less than the memory access time, so the TCAM's access time would limit lookup throughput. To increase the lookup throughput, the architecture could select more than one segment in the TCAM and simultaneously perform multiple IP lookups. This is possible by duplicating the decoder-PEB logic in level 1 and resolving additional IP prefixes in the TCAM array. Similarly, concurrent lookup and update are also feasible using such architecture. However, the more TCAM segments the lookup enables, the more power the TCAM consumes, so a designer must trade off power consumption and throughput.

Empirical model for memory and power

We now present empirical formulas to compute the total memory requirement and power consumption of our proposed architecture. Let $rows_i$ represent the number of entries in the ith TCAM segment; and *tagbits*, the TCAM word length (24 bits). The proposed architecture's minimum memory requirement (for 32-bit entries) is

$$\sum_{i=1}^{256} (rows_i \times tagbits / 32) \quad (1)$$

We base our empirical model for power estimation on the number of entries enabled during the lookup process. Based on this model, the power consumption (for 32-bit entries) in a TCAM with its ith segment enabled is

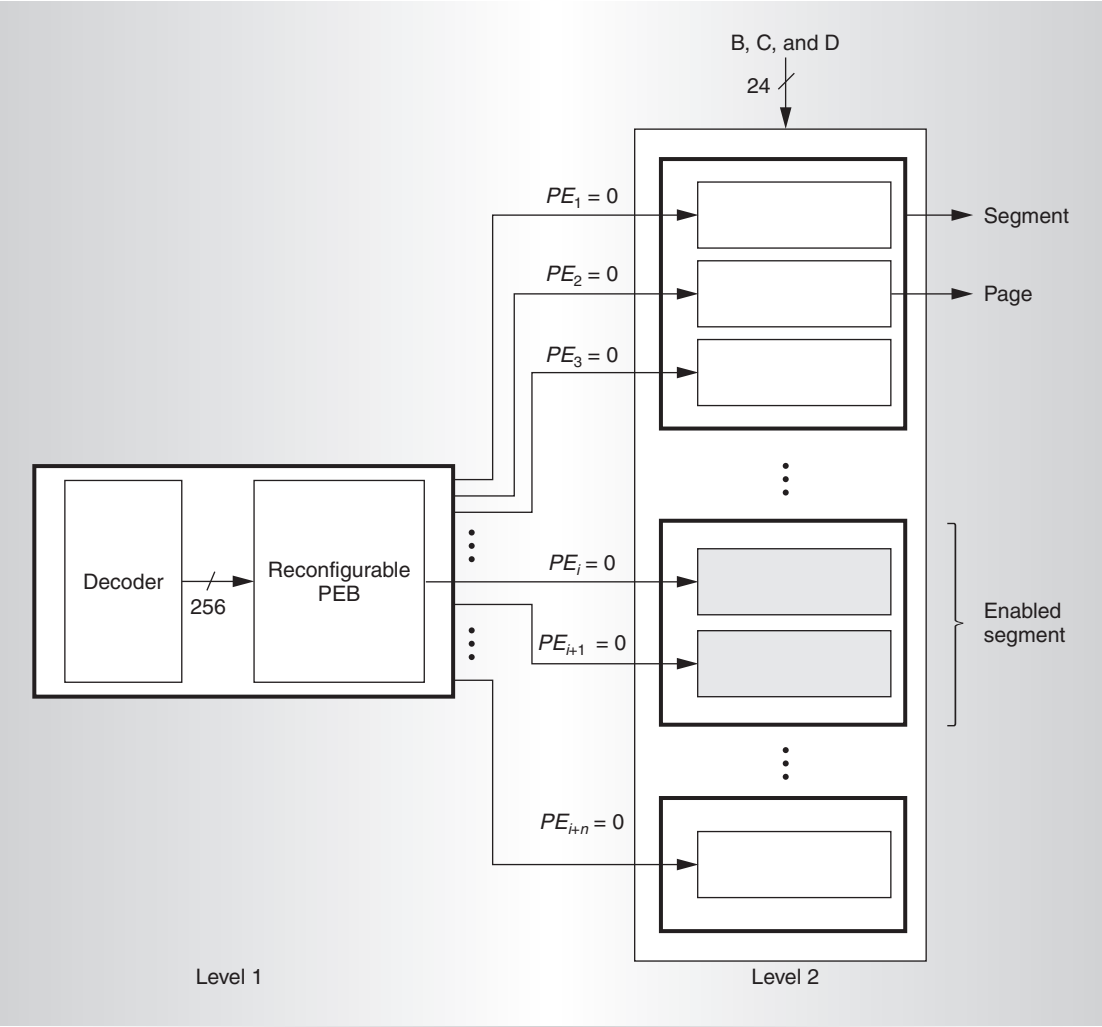

Figure 5. TCAM architecture for routing lookup.

$$P(rows, tagbits) = k \times (rows_i \times tagbits/32) \quad (2)$$

where k is the power constant.

The largest segment in the TCAM array determines the maximum power consumption for the lookup process.

Fast incremental update

Approximately 100 to 1,000 updates per second take place in core routers today.[13] Thus, the update operation should be incremental and fast to avoid becoming a bottleneck in the search operation.

A router receives information—routing updates—about the route changes from its nearby routers. During each routing update (an insertion or deletion), we apply compaction techniques to avoid any redundant storage in the TCAM. However, it is possible that more than one prefix in the TCAM will need updating, a situation called TCAM update.[14] In Liu's technique, for a routing update, compaction would require the minimization of about 10,000 prefixes at a time. Minimizing such a large set of prefixes introduces two types of delays: The first delay comes from Espresso-II's computation time. The second delay comes from TCAM entry updates. If M is the size of the prefix set to be minimized, then it takes $O(M)$ time for a TCAM insert operation. This is because the TCAM must scan the minimized set of prefixes and update only those prefixes that have changed. Deleting an entry from the TCAM is even costlier, taking $O(M^2)$ time. In our technique, because the maximum value of M

Table 1. Compaction using prefix overlapping.

Router	No. of prefixes: Before compaction	After compaction	Compaction (percentage)
is	801	648	19.1
utah	145	133	8.3
attcanada	112,412	79,769	29.0
bbnplanet	124,538	92,774	25.5

is 256 (rather than 10,000), the insertion and deletion steps are fast and truly incremental.

For any routing update, our approach restricts TCAM updates to a segment. So it is possible to update several segments simultaneously by modifying the decoder-PEB unit. Our technique can achieve a higher number of updates per second than what a single TCAM chip supports by using several TCAM chips and placing each on a separate bus.

In our architecture, it is possible for the segment size to grow during update, possibly leading to an overflow. Rebuilding the entire routing table would take care of the overflow. However, a rebuild is time consuming, so unsuitable for fast lookup. Providing empty pages in the TCAM array will help manage the overflow. Reconfiguring the PEB on the fly will permit an empty page to become part of the overflowing segment. This reconfiguration takes microseconds.

Experimental setup and results

We simulated our approach using a chipset containing two 1.6-GHz AMD Athlons to compute the prefix minimizations. The prefix traces for minimization came from four routers (`is`, `utah`, `bbnplanet`, and `attcanada`), and we evaluated memory compaction, minimization time, and worst-case power consumption. For memory compaction and minimization time, we compared our approach to Liu's.[3] We also compare our worst-case power consumption to the results of Liu;[3] Zane, Narlikar, and Basu;[1] and Panigrahy and Sharma.[2]

Memory compaction

Table 1 indicates the compaction achieved by prefix overlapping. For example, the case of `attcanada` shows a 29-percent compaction.

Table 2 shows the compaction from our prefix minimization approach, and compares it to Liu's results. The second column shows the number of prefixes remaining if we were to minimize as many prefixes as possible in the given routing table. Our approach and Liu's group the prefixes before minimization; columns three and four show the number of prefixes remaining after minimization, for each approach. Although neither approach eliminates all the prefixes possible, each significantly compacts the routing tables.

However, for large routing tables, we could not determine the prefix minimization of Liu's approach, even after running the simulation for two days. Also, our approach compacts the smaller routing tables of `is` and `utah` more than Liu's approach, because our approach uses prefix expansion. In fact, our approach achieves more than 33 percent compaction for the `attcanada`, `bbnplanet`, and `is` routers, as column five of Table 2 shows.

We also determined the compaction of both

Table 2. Compaction using prefix minimization.

Router	No. of prefixes: After minimizing all possible prefixes	After prefix minimization (our approach)	After prefix minimization (Liu's approach)	Compaction of our approach (percentage)
is	434	505	562	36.9
utah	123	123	132	15.2
attcanada	Hangs*	70,368	Hangs*	37.4
bbnplanet	Hangs*	83,468	Hangs*	33.0

* The minimization processes do not complete within two days.

Table 3. Compaction using prefix minimization and overlapping.

Router	No. of prefixes: Our approach	No. of prefixes: Liu's approach	Our approach's compaction (percentage)	Improvement (percentage)
is	348	460	56.6	24.7
utah	87	123	40.0	62.1
attcanada	43,378	54,476	61.4	15.2
bbnplanet	53,625	69,646	56.9	22.6

Table 4. Timing analysis for minimization.

Router	Our approach: Time (seconds)	Our approach: No. of prefixes updated	Liu's approach: Time (seconds)	Liu's approach: No. of prefixes updated
is	0.003	34	0.159	373
utah	0.002	8	0.008	44
attcanada	0.005	143	1,098.47	15,146
bbnplanet	0.006	171	63.04	7,580

approaches when they include prefix overlapping followed by prefix minimization. Column 2 of Table 3 shows the number of entries remaining after our approach minimizes every prefix possible; we compute this number using equation 1. Column 3 shows the results that Liu reported earlier. Column 4 indicates our approach's total routing table compaction, and column 5 shows the percentage improvement in compaction over Liu's results. Our approach offers significant improvements because it exploits the compaction opportunities inherent in our architecture and the prefix properties.

Minimization time

Table 4 shows the runtime for minimizing the prefixes using Espresso-II in both approaches. The results indicate the maximum possible time that the 1.6-GHz Athlon, dual-processor chipset takes to minimize the prefixes when the neighboring router requests the addition of a new prefix. The worst-case minimization time for Liu's approach was 1,098.47 seconds for the **attcanada** router (15,146 prefixes), which is very expensive for incremental updates. Our approach, on the other hand, took a worst-case time of 0.006 seconds for **bbnplanet**. This value is not only small and practical for real-life updates, but also bounded because at any point of time the number of inputs to Espresso-II will never exceed 256. Espresso-II's runtime also increases with the number of bits in the prefixes under consideration. Our approach uses eight bits as opposed to the 32 bits for Liu's approach, further reducing runtime.

Worst-case power consumption

TCAM power consumption is proportional to the number of bits enabled in the TCAM during the search operation.[1] Table 5 shows the worst-case power consumption (32-bit equivalent) for several approaches. Liu's approach enables the entire TCAM (N prefixes) during every lookup operation, making it very costly. Panigrahy and Sharma's approach enables a large number of entries (16,000)[2] and 8,000 group IDs. In the bitmap technique, although the number of entries enabled during search (C_{max}) is very small, the entries with prefixes of less than 16 and greater than 24 bits are always enabled during every lookup operation. The number of these entries, P_b, is very large (65,068) for the **bbnplanet** router. The partition technique uses an index TCAM and a data TCAM. It enables only a page of size N/b in the data TCAM, where b is the number of pages in the TCAM. For every lookup, however, this technique searches the entire index TCAM of size S_d, which is large if the page size is small. When the page size is large, the index

Table 5. Worst-case bounds on the number of enabled entries.

Approach	Worst-case no. of enabled entries
Liu	$O(N)$
Panigrahy and Sharma	$O(16{,}000 + p \times 6/32)$
Bitmap	$O(C_{max} + P_b)$
Partition	$O(N/b + S_d)$
Proposed	$O(S_i)$

Table 6. Relative power consumption per lookup.

Router	No. of enabled bits	Total no. of bits in the TCAM	Savings (percentage)
is	205×24	460×32	67
utah	17×24	123×32	90
attcanada	$4{,}582 \times 24$	$544{,}476 \times 32$	99
bbnplanet	$5{,}596 \times 24$	$69{,}646 \times 32$	94

TCAM has few entries; however, it is the size of the page enabled in the data TCAM that makes the main contribution to power consumption. Our approach enables only a segment (S_i) of the TCAM, which is quite small compared to the entire TCAM. For example, in the **bbnplanet** router, the enabled segment has about 5,600 prefixes. Because most segments in our TCAM are very small, the average power consumption is quite small. It might be important to mention that the word length of the TCAM in our architecture is 24 bits as opposed to the 32 bits in conventional TCAMs.

Table 6 shows the worst-case power consumption per lookup in terms of the number of bits enabled; we calculate this value using equation 2. Our approach saves significant power compared to approaches that enable entire TCAMs.

The proposed architecture enables only a small portion of TCAM array during the lookup operation and achieves significant power savings. Results indicate that such an architecture holds promise in supporting storage- and energy-efficient router designs. We plan to further reduce the update time to increase the throughput of incremental updates using a new approach on minimization (other than Espresso-II) that will need reduced minimization time. MICRO

References

1. F. Zane, G. Narlikar, and A. Basu, "CoolCAMs: Power-Efficient TCAMs for Forwarding Engines," *Proc. IEEE Infocom 2003*, IEEE Press, 2003, pp. 42-52.
2. R. Panigrahy and S. Sharma, "Reducing TCAM Power Consumption and Increasing Throughput," *Proc. 10th Symp. High-Performance Interconnects* (HOTI 02), IEEE CS Press, 2002, p. 107-112.
3. H. Liu, "Routing Table Compaction in Ternary CAM," *IEEE Micro*, Jan.-Feb. 2002, vol. 22, no. 1, pp. 58-64.
4. M.J. Akhbarizadeh and M. Nourani, "An IP Packet Forwarding Technique Based on Partitioned Lookup Table," *Proc. IEEE Int'l Conf. Comm.* (ICC 02), IEEE Press, 2002, pp. 2263-2267.
5. R.K. Brayton et al., *Logic Minimization Algorithms for VLSI Synthesis*, Kluwer Academic Publishers, 1984.
6. S. Nilsson and G. Karlsson, "IP-Address Lookup Using LC-Tries," *IEEE J. Selected Areas Comm.*, vol. 17, no. 6, June 1999, pp. 1083-1092.
7. M. Waldvogel et al., "Scalable High-Speed IP Routing Table Lookups," *Computer Comm. Rev.*, vol. 27, no. 4, Oct. 1997, pp. 25-36.
8. P. Gupta, *Algorithms for Routing Lookups and Packet Classification,* doctoral dissertation, Dept. Computer Science, Stanford Univ., 2000.
9. P. Gupta, S. Lin, and N. McKeown, "Routing Lookups in Hardware at Memory Access Speeds," *Proc. IEEE Infocom 1998*, IEEE Press, 1998, pp. 1240-1247.
10. M. Kobayashi, T. Murase, and A. Kuriyama, "A Longest Prefix Match Search Engine for Multigigabit IP Processing," *Proc. IEEE Int'l Conf. Comm.* (ICC 00), IEEE Press, 2000, pp. 1360-1364.
11. S. Sikka and G. Varghese, "Memory-Efficient State Lookups with Fast Updates," *Proc. ACM Special Interest Group on Data Comm.* (SIGCOMM 00), ACM Press, 2000, pp. 335-347.
12. V. Srinivasan and G. Varghese, "Fast Address Lookups Using Controlled Prefix Expansion," *ACM Trans. Computer Systems*, vol. 17, no. 1, Oct. 1999, pp. 1-40.
13. C. Labovitz, G.R. Malan, and F. Jahanian,

"Internet Routing Instability," *IEEE/ACM Trans. on Networking*, vol. 6, no. 5, Oct. 1998, pp. 515-528.
14. D. Shah and P. Gupta, "Fast Updating Algorithms for TCAMs," *IEEE Micro,* vol. 21, no. 1, Jan.-Feb. 2001, pp. 36-47.

Ravikumar V.C. is a system designer at Hewlett-Packard. His research interests include networking, operating systems, and embedded systems. He has an MS degree in computer science from Texas A&M University.

Rabi N. Mahapatra is an associate professor with the Department of Computer Science at Texas A&M University. His research interests include embedded-system codesign, system on chips, VLSI design, and system architectures. Mahapatra has a PhD from the Indian Institute of Technology, Kharagpur. He is a senior member of the IEEE Computer Society.

Direct questions and comments to Rabi N. Mahapatra, Dept. of Computer Science, Texas A&M Univ., College Station, TX 77845; rabi@cs.tamu.edu.