# IPStash: A Set-Associative Memory Approach for Efficient IP-lookup

Stefanos Kaxiras
*Department of Electrical and Computer Engineering*
*University of Patras, Greece*
*kaxiras@ee.upatras.gr*

Georgios Keramidas
*Department of Electrical and Computer Engineering,*
*University of Patras, Greece*
*keramidas@ee.upatras.gr*

***Abstract*—IP-Lookup is a challenging problem because of the increasing routing table sizes, increased traffic, and higher speed links. These characteristics lead to the prevalence of hardware solutions such as TCAMs (Ternary Content Addressable Memories), despite their high power consumption, low update rate, and increased board area requirements. We propose a memory architecture called IPStash to act as a TCAM replacement, offering at the same time, high update rate, higher performance, and significant power savings. The premise of our work is that full associativity is not necessary for IP-lookup. Rather, we show that the required associativity is simply a function of the routing table size. Thus, we propose a memory architecture similar to set-associative caches but enhanced with mechanisms to facilitate IP-lookup and in particular longest prefix match (LPM). To reach a minimum level of required associativity we introduce an iterative method to perform LPM in a small number of iterations. This allows us to insert route prefixes of different lengths in IPStash very efficiently, selecting the most appropriate index in each case. Orthogonal to this, we use skewed associativity to increase the effective capacity of our devices. We thoroughly examine different choices in partitioning routing tables for the iterative LPM and the design space for the IPStash devices. The proposed architecture is also easily expandable. Using the Cacti 3.2 access time and power consumption simulation tool we explore the design space for IPStash devices and we compare them with the best blocked commercial TCAMs.**

**Keywords** (*Network architectures, Network routers, routing table lookup, Ternary Content Addressable Memories, set-associative memories)*

## I. INTRODUCTION

Independently of a router's Internet hierarchy level —core, edge, or access platform— a function that must be performed in the most efficient manner is packet forwarding. In other words, determining routing, security and QoS policies for each incoming packet based on information from the packet itself. A prime example is the Internet Protocol's basic routing function (IP-lookup) which determines the next network hop for each incoming packet. Its complexity stems from wildcards in the routing tables, and from the Longest Prefix Match (LPM) algorithm mandated by the Classless Inter-Domain Routing (CIDR).

Since the advent of CIDR in 1993, IP routes have been identified by a <*route prefix, prefix length*> pair, where the prefix length is between 1 and 32 bits. For every incoming packet, a search must be performed in the router's forwarding table to determine the packet's next network hop. The search is decomposed into two steps. First, we find the set of routes with prefixes that match the beginning of the incoming packet's IP destination address. Then, among this set of routes, we select the one with the longest prefix. This identifies the next network hop.

What makes IP-lookup an interesting problem is that it must be performed increasingly fast on increasingly large routing tables. One direction to tackle this problem concentrates on partitioning routing tables in optimized data structures, often in tries (digital trees), so as to reduce as much as possible the average number of accesses needed to perform LPM [2,17,19,26]. Each lookup however, requires several (four to six) dependent-serialized memory accesses stressing conventional memory architectures to the limit. Memory latency and not bandwidth is the limiting factor with these approaches. Significant effort has been devoted to solve the latency problem either by using fast RAM (e.g., Reduced Latency DRAM—RLDRAM) or by replicating the routing table over several devices so that searches can run in parallel to attain the necessary speeds [3]. The first solution can only mitigate the problem and the second solution drives up system costs (due to bus replication) and further complicates routing table update. In all cases the solution is a trade-off among search speed, update speed and memory size.

***TCAMs*—**A fruitful approach to circumvent latency restrictions is through parallelism: searching all the routes simultaneously. Content Addressable Memories perform exactly this fully-parallel search. To handle route prefixes, Ternary CAMs (TCAMs) are used which have the capability to represent wildcards. TCAMs have found acceptance in many commercial products; several companies (IDT [7], Netlogic [16], Micron [15], Sibercore [25]) currently offer a large array of TCAM products used in IP-lookup and packet classification.

In a TCAM, IP-lookup is performed by storing routing table entries in order of decreasing prefix lengths. TCAMs automatically report the first entry among all the entries that match the incoming packet destination address (topmost match).

The need to maintain a sorted table in a TCAM makes incremental updates a difficult problem. If $N$ is the total number of prefixes to be stored in an $M$-entry TCAM, naive addition of a new update can result in $O(N)$ moves. Significant effort has been devoted in addressing this problem [9,24], however all the proposed algorithms require an external entity to manage and partition the routing table.

In addition to the update problems, two other major drawbacks plague TCAMs: high cost/density ratio and high power consumption. The fully-associative nature of the TCAM means that comparisons are performed on the whole memory array, costing a lot of power: a typical 18 Mbit 512K-entry TCAM can consume up to 15 Watts when all the entries are searched [7,25]. TCAM power consumption is critical in router applications because it affects two important router characteristics: *linecard power* and *port density.* Linecards have fixed power budgets because of cooling and power distribution constraints [5]. Thus, one can fit only a few power-hungry TCAMs per linecard. This

in turn reduces port density —the number of input/output ports that can fit in a fixed volume— increasing the running costs for the routers.

Efforts to divide TCAMs into "blocks" and search only the relevant blocks have reduced power consumption considerably [7,16,18,21,29,30]. This direction to power management actually validates our approach. "Blocked" TCAMs are in some ways analogous to set-associative memories but in this paper we argue for pure set-associative memory structures for IP-lookup: many more "blocks" with *less associativity* and *separation* of the comparators from the storage array. In TCAMs, blocking further complicates routing table management requiring not only correct sorting but also correct partitioning of the routing tables. Routing table updates also become more complicated. In addition, external logic to select blocks to be searched is necessary. All these factors further increase the distance between our proposal and TCAMs in terms of ease-of-use while still failing to reduce power consumption below that of a straightforward set-associative array.

More seriously, blocked TCAMs can only reduce average power consumption. Since the main constrain in our context is the fixed power budget of a linecard a reduction of average power consumption is of limited value —maximum power consumption still matters. As we show in this paper, the *maximum* power consumption of IPStash is less than the power consumption of a comparable blocked TCAM with full power management.

***IPStash*—**To address TCAM problems we propose a new memory architecture for IP-lookup we call *IPStash*. It is based on the simple hypothesis that IP-lookup only needs associativity depending on routing table size; *not full associativity* (TCAMs) or *limited associativity* ("blocked" TCAMs). As we show in this paper this hypothesis is indeed supported by the observed structure of typical routing tables. IPStash is a set-associative memory device that directly replaces a TCAM and offers at the same time:

- Better functionality: It behaves as a TCAM, i.e., stores the routing table and responds with the longest prefix match to a single external access. In contrast to TCAMs there is no need for complex sorting and/or partitioning of the routing table; instead, a simple route-prefix expansion is performed but this can happen automatically and transparently.
- Fast routing table updates: since the routing table needs no special handling, updates are also straightforward to perform. Updates are simply writes/deletes to/from IPStash.
- Low power: Accessing a set-associative memory is far more power-efficient than accessing a CAM. The difference is accessing a very small subset of the memory and performing the relevant comparisons, instead of accessing and comparing the whole memory at once.
- Higher density scaling: One bit in a TCAM requires 10-12 transistors while SRAM memory cells require 4-6 transistors. Even when TCAMs are implemented using DRAM technology they can be less dense than SRAMs.
- Easy expandability: Expanding the IPStash is as easy as adding more devices in parallel without the need for any complicated arbitration. The net effect is an increase of the associativity of the whole array.
- Error Correction Codes: The requirement for ECC is fast becoming a necessity in Internet equipment. Intergrating ECC in IPStash (SRAM) is as straightforward as in set-associative caches but as of yet it is unclear how ECC can be efficiently implemented in TCAMs. In the latter case, all memory must be checked for errors on every access since it is impossible to tell a no-match from a one-bit error.

***Contributions of this paper*—**The contributions of this paper are as follows:

- We propose a set-associative memory architecture enhanced with the necessary mechanisms to perform IP-lookup. Furthermore, we introduce an iterative method to perform Longest Prefix Match which results in very efficient storage of the routing tables in set-associative arrays. In addition, we show how *skewed associativity* can be applied with great success to further increase the effective capacity of IPStash devices.
- We exhaustively search the design space in two dimensions. First we examine the choices on how to partition routing tables for the iterative longest prefix match. The partitioning affects how efficiently the routing tables can fit in an IPStash. Second, we examine the design space of IPStash devices showing the trade-off between power consumption and performance.
- We introduce a power optimization that takes advantage of the iterative nature of our LPM search and selectively powers-down set-associative ways that contain irrelevant entries.
- We use real data to validate our assumptions with simulations. We use the Cacti tool to estimate power consumption and performance and we show that IPStash can be up to 64% more power efficient or 160% faster than the best commercial available blocked TCAMs.

Compared to our earlier proposal [8] for a set associative memory for IP-lookup: i) we have resolved its major shortcoming which was the significant expansion of the route prefixes (which resulted in expanded routing tables twice their original size), ii) we introduce a new power-management technique leading to new levels of power-consumption efficiency and iii) while our earlier work concerned a specific point in the design space of set-associative memories for IP-lookup, in this paper we systematically explore a much larger space of possible solutions.

***Structure of this paper*—**Section II presents the IPStash architecture and our implementation of the LPM algorithm. In Section III we show that IP-lookup needs associativity depending on the routing table size. Section IV presents other features of the architecture. Section V provides simulation results for power consumption and Section VI discusses related work. Finally, Section VII offers our conclusions.

## II. IPSTASH ARCHITECTURE

The main idea of the IPStash is to use a set-associative memory structure to store routing tables. IPStash functions and looks like a set-associative cache. However, in contrast to a cache which holds a small part of the data set, IPStash is intended to hold a routing table in its entirety. In other words, it is the main storage for the routing table—*not a cache for it.*In this section we describe how routing tables can be inserted in a set-associative structure and how LPM is performed in this case.

### *A IPStash Basics*

To insert routing prefixes in a set-associative structure —as opposed to a TCAM— we first need to define an index. Routing prefixes can be of any length but in reality there are no prefixes

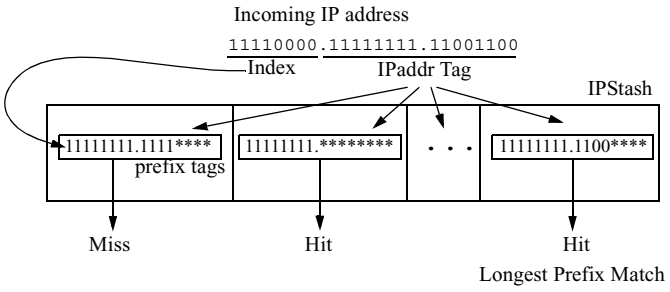


Fig. 1. IPStash with variable prefix tags

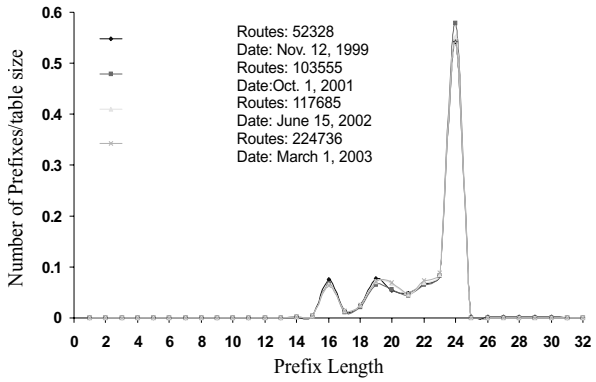


Fig. 2. Prefix length distribution (from 1999 to 2003)

shorter than 8 bits. Thus, we can count on at least the 8 most significant bits as the index. Disregarding for a moment the inefficiency of such an indexing scheme, let us assume that we do insert routing prefixes in a set-associative structure using their 8 leftmost bits (most significant positions) as index. To retrieve a prefix from IPStash we also need a tag. Any non-wildcard bits beyond the 8 leftmost index bits then comprise the tag. Tags are variable in IPStash: 0 to 24 bits with an 8-bit index. The prefix length, stored with the tag, either as binary value or as a mask, defines the length of the tag and how many bits participate in the tag match. Fig. 1 shows a set of a set-associative array containing several prefix entries with different lengths. An incoming IP address can match many of them as in a TCAM. Viewed differently, the variable tag match provides the same functionality as the TCAM wildcards. The key observation here is that routing prefixes have their wildcard bits always bundled together in the right side (least significant positions) affording us variable tags and easy implementation of variable-tag match.

Of course, to perform LPM we need to select the longest of all the matching prefixes in a set. To do this we need another level of *length arbitration* after the tag match that gives us the longest matching prefix. Again, the prefix length, stored with the matching tags, is used in comparisons to select the longest prefix. If the prefix length is stored as a binary value it is expanded into a full bit mask. The maximum length can be found by comparing the masks with a combinatorial circuit or using a *length arbitration* bus with as many lines as the maximum prefix length. Arbitration works as follows: When multiple tags match simultaneously, they assert the wire that corresponds to their prefix length. Every matching tag sees each other's length and a self-proclaimed winner outputs its result on the output bus. All other matching tags withdraw.

As mentioned above an 8-bit index and especially the MSB bits would be disastrous for the associativity requirements for a large routing table. Conflict chains would be unacceptably long. In the next subsections we show two things. First, how we can increase the index to address a larger number of sets. Second, how we can partition the routing table into classes, each with its own index, to dramatically increase the efficiency of storing a routing table in a set-associative array. Both of these techniques are driven by the structure of the routing tables which we analyze next.

### *B Routing Table Characteristics*

Many researchers have observed a distinct commonality in the distribution of prefix lengths in routing tables [17,19,26] that stems from the allocation of IP addresses in the Internet as a result of CIDR. This distribution is *not* expected to change significantly with time [6]. Fig. 2 shows the distribution of prefix lengths for four tables taken from [22] and from different time periods (from 1999 to 2003). We can easily draw some general conclusions —also noted by other researchers— from the graphs in Fig. 2: the distribution is the same for all tables regardless of their size and creation date. With respect to the actual prefix lengths: 24-bit prefixes comprise about 60% of the tables; prefixes longer than 24 bits are very few (about 1%); there are no prefixes less than 8 bits; the bulk (about 97%) of the prefixes have lengths between 16 and 24 bits.

### *C Prefix expansion and index selection*

A straightforward method to increase the index is to use a controlled prefix expansion technique to expand prefixes to larger lengths. For example, we can expand prefixes of lengths 8,9,10, and 11 all to length 12 thus having the opportunity to use up to 12 bits as index.

The controlled prefix expansion creates comparably very few additional expanded prefixes at these short lengths simply because they are very few short prefixes to begin with. This, however, is not true for all prefix lengths as it can be seen in Fig. 2. As we expand prefixes into larger and larger lengths, routing-table inflation becomes a significant problem.

Unfortunately, it is desirable to expand prefixes to large lengths in order to gain access to the "best" indexing bits. Fig. 3 shows the *bit entropy* for prefixes of length 16 to 20 (upper graph) and 21 to 24 (lower graph). The y-axis is the prefix length, the x-axis represents the bits (up to bit 24), and the z-axis is the entropy of the bits. Bit entropy is the bit's apparent randomness —how un-biased it seems towards one or zero. The higher the entropy the better the bit for indexing. Indexing with high entropy bits will help to spread references more evenly across the memory minimizing the associativity requirements. MSB bits have very low entropy and are really unsuitable for indexing. Regardless of prefix length, the best bits for indexing start from bit 6 and reach the prefixes' maximum length.

The above analysis suggests expansion of prefixes to large lengths and selection of the right-most (non-wildcard) bits as index —prefix expansion creates high entropy bits. Even if we could accept routing-table inflation, prefix expansion alone is not sufficient for efficient storage of a routing table into a set-associative structure —even with a very good index, a single hashing of the routing table still results in unacceptably large associativity.

### *D Class Partitioning and Iterative LPM*

To address this problem, we introduce an iterative LPM where we search for progressively shorter prefixes. This allows

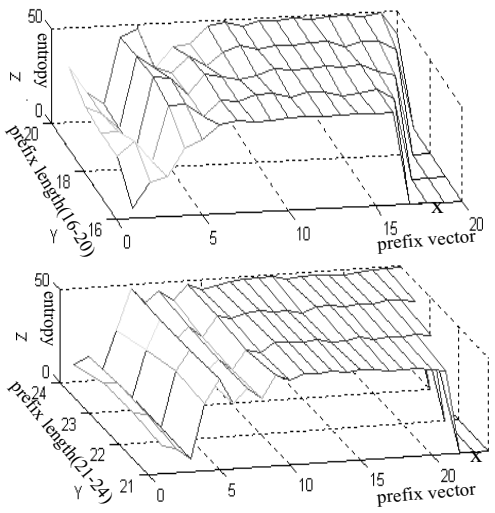


Fig. 3. Bit entropy for prefixes 16- to 24-bits long (based on 8 routing tables)

us to treat each prefix length independently of all others. Thus, we can insert, for example, prefixes of length 32 into IPStash using the most appropriate index; similarly, we insert prefixes of length 31,30,29,..., using again the most appropriate index from the available non-wildcard bits. To perform LPM we start by searching the longest prefixes using their corresponding index to retrieve them. We repeat with progressively shorter prefix lengths until we find the first match —the LPM.

But iterating over 24 prefix lengths (lengths 32 to 8) is impractical. First, it would make some searches unacceptably slow if we had to try several different lengths until we found a match. Second, it would introduce great variability in the hit latency which is clearly undesirable in a router/network processor environment.

Our solution is to partition prefixes into a small set of classes and iterate over the classes. For example, we can partition the routing table into the following classes:

- **Class 1** contains all the prefixes from 21 to 32 bits. Any 12 (or any other number if we chose so) of the first 21 bits can be used for indexing —bits above 21 are wildcard bits.
- **Class 2** contains all the prefixes from 17 to 20 bits. Any 12 bits of the first 17 can be used as an index, but bits 18 to 20 contain wildcards.
- **Class 3** contains all the prefixes from 8 to 16 bits. Only this class —the last class containing the shortest prefixes— requires prefix expansion of the shorter prefixes to guarantee the availability of the index bits.

Class partitioning is nothing more than a definition of the index (consequently of the tag) for a set of prefix lengths. It allows us to *re-hash* a routing table multiple times, each hash using an optimal index. Fig. 4 shows the associativity requirements for 8 routing tables when they are single-hashed (single class), doubly-hashed (2 classes) and triply-hashed (3 classes). The benefit from more than 3 classes is little; we have not seen significant improvement going from 3 to 4 classes. The optimal class partitioning depends on the actual routing table to be stored and can change over-time. Thus, IPStash is configurable with respect to the classes used to store and access a routing table.

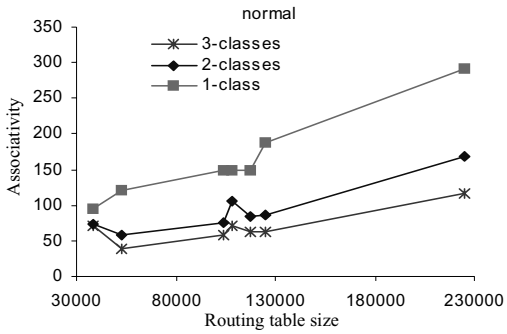


Fig. 4. Associativity requirements with 1, 2, and 3 classes for 8 routing table

### *E A working example*

To put it all together Fig. 5 shows how the index and tag are extracted from a prefix belonging to some class. The class boundaries define the range of prefix lengths that belong to the class. The lower class boundary guarantees that no bit below that boundary can be a wildcard bit for the prefixes belonging to the specific class. Thus, the index can always be safely chosen from bits below the lower class boundary. Any bits below the lower class boundary besides the index bits form the *fixed tag* of the prefix while non-wildcard bits above the lower class boundary form the *variable* part of the prefix tag. The length of the prefix is used to form a mask that controls exactly how many bits of the tag participate in the tag match. This mask is stored with the tag in each entry.

To insert a prefix in IPStash we first assign it to a class, extract its index and form its tag by concatenating its fixed tag parts with the variable part. In the same time we form the mask stored with the tag that controls tag match (Fig. 6).

To perform LPM in IPStash we iteratively search all classes until we find a match. For each class we take the incoming IP address, extract the class index and form the corresponding tag to be compared against the stored prefix tags (Fig. 7). The IP address tag is a full tag containing all the IP address bits but when it is compared to the stored prefix tags the corresponding masks control which bits participate in the comparison and which bits are ignored (Fig. 7).

### *F Skewed associativity*

Although there are significant gains going from a single hash (single class) of the routing table to wo and three hashes (2 and 3 classes) —possibly accompanied by a prefix expansion to secure an index for the shortest class— Fig. 4 shows that there are still considerable associativity requirements even for triple-hashing. Our second proposal, *orthogonal* to class partitioning, for increasing hashing effectiveness and decreasing associativity requirements is based on Seznec's idea of a skewed associativity [23]. Skewed associativity can be applied in IPStash with great success. The basic idea of skewed associativity is to use different indexing functions for each of the set-associative ways. Thus, items that in a standard cache would compete for a place in the same set because of identical indexing across the ways, in a skewed-associative cache map on different sets. One way to think about skewed associativity is to view it as an additional increase of the *entropy* of the system by the introduction of additional randomness in the distribution of the items in the cache.

The left upper graph of Fig. 8 shows how RT5 is loaded into an "unlimited-associativity" IPStash using 12 bits for index and the three class approach—without restriction to the number

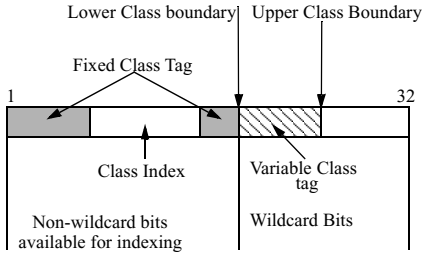


Fig. 5. Index and Tag of a prefix

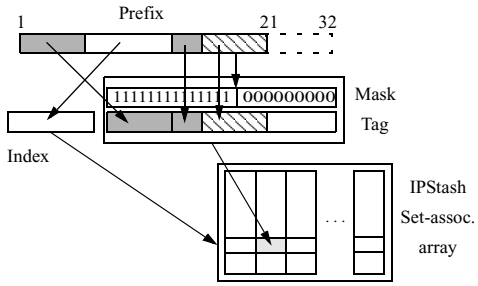


Fig. 6. Prefix insertion into IPStash

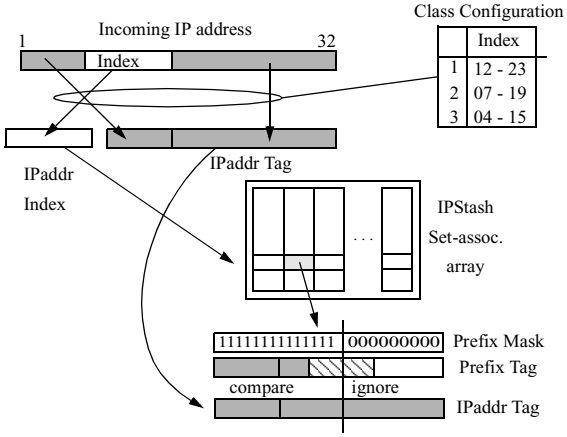


Fig. 7. Tag match in IPStash

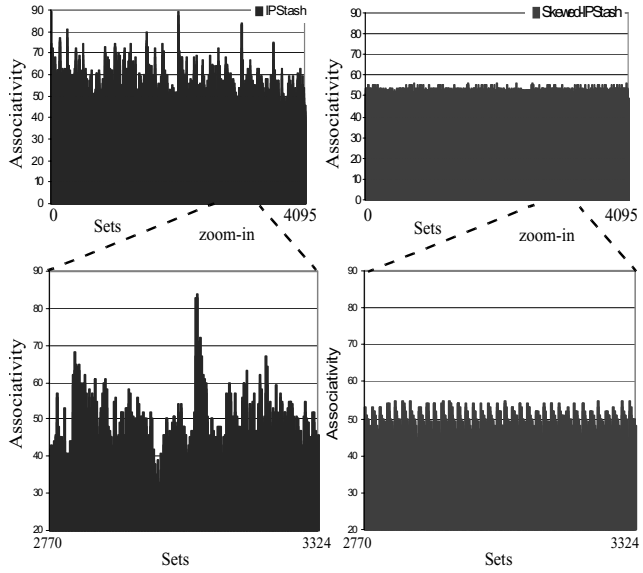


Fig. 8. Original IPStash and skewed IPStash

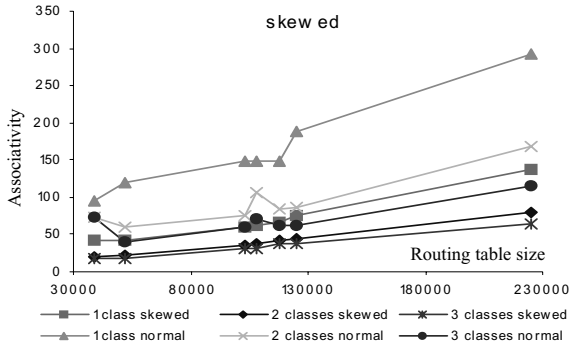


Fig. 9. Skewed-Associativity requirements with 1, 2, and 3 classes for 8 routing tables

of ways. The horizontal dimension represents the sets and the vertical dimension the set-associative ways. As it is depicted in the graph, RT5 needs anywhere from 23 to 89 ways. If RT5 was forced into a 64-way IPStash anything beyond 64 in the graph would be a conflict. Despite the random look of the graph, the jagged edges do in fact represent order (structure) in the system. It is the order introduced by the hashing function. The effect of skewing (shown in the right graph of Fig. 8) is to smooth-out the jagged edges of the original graph.

We use a simple skewing technique, XORing index bits with tag bits rotated once for each new skewed index. Details can be found in [8]. Because many a time we do not have enough available tag bits we create only a few distinct skewed indices regardless of the hardware associativity and apply each skewed index to multiple ways. Although this technique might not give us optimal results it has the desirable characteristic of curbing the increase in power consumption due to the multiple distinct decoders.

The effect of skewed associativity is shown in Fig. 9 which compares the associativity requirements with and without skewing and for 1,2, and 3 classes for all 8 routing tables. The benefits are significant across all cases, comparable and additive to the benefits from multiple hashing. A distinct effect of skewing is to "linearize" the required associativity curves and bring them very close to the best possible outcome as it is further analyzed in Section III.

## III. Detailed Analysis of Memory Requirements

Up until now we have discussed required associativity as a function of the routing table size. In this section we examine the memory overhead when we try to fit a routing table into a fixed-associativity IPStash. A significant difference between IPStash and a TCAM is that the TCAM can fit a routing table with exactly the same number of entries as its nominal capacity, while IPStash has some inherent capacity inefficiencies due to imperfect hashing. The inefficiencies are divided into two kinds:

- Inefficiency stemming from the increased size of the routing tables because of prefix expansion in the shortest class to

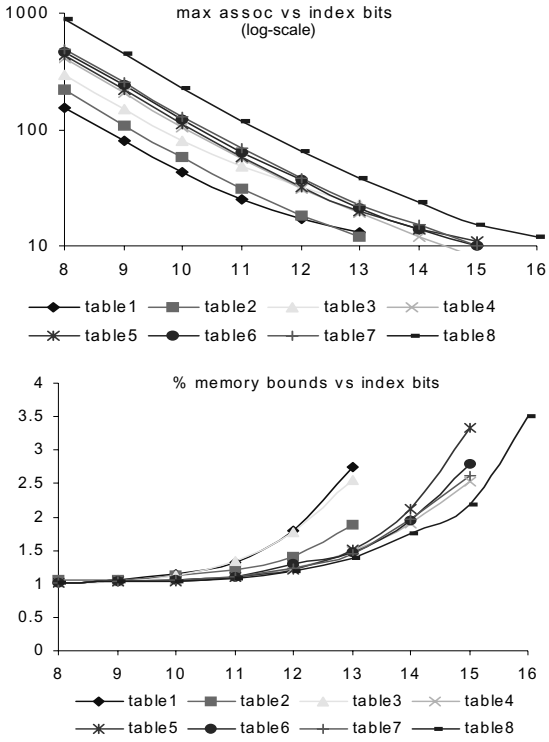


Fig. 10. Memory bounds and max associativity vs index bits for all the tables

secure a desired index.

- Inefficiency stemming from imperfect hashing of the routing tables. Assuming that IPStash's associativity equals the routing table's required associativity, this inefficiency is nothing else than the empty slots left in the sets where the associativity is less than maximum.

Our approach to assess memory overhead in IPStash is to exhaustively study the choices for different indices and class configurations per index. We examine several different index lengths from 8 to 16 bits. For a given index, we select a class configuration, which —for simplicity— is common to all 8 routing tables we use. We have also examined class configurations tailored individually for each routing table which gives us a small additional benefit. Imbedded in the class configuration is the prefix expansion in the shortest class. Fig. 10 shows the normalized memory overhead (lower part) and required associativity (upper part) for all the tables used in this paper. In all cases, the class configuration that minimizes the average memory overhead of the 8 routing tables is shown.

Detailed results are presented in Table I which shows the effect of the index on the number of the expanded prefixes and on the memory overhead (for both skewed and non-skewed cases). Fig. 10 and Table I show that as the number of index bits grows, memory overhead is increasing and the required associativity is decreasing. In both cases, the trends are exponential. On one hand we are seeking low associativity for an efficient implementation of IPStash. On the other, increasing the index to decrease associativity, increases both capacity inefficiencies of IPStash: we have to both store larger expanded tables and the empty slots left in sets correspond to a larger percentage of wasted memory in low associativity.

This is clear in Fig. 11 which shows the relationship of the

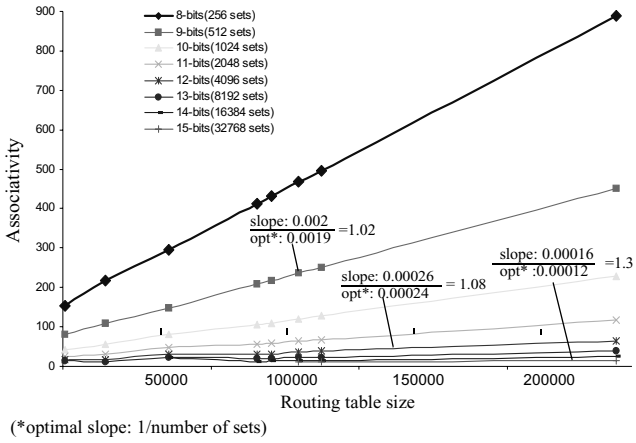


Fig. 11. Associativity and Routing table size

required associativity (skewed case) to the initial size for our eight routing tables. As we can see this relationship is remarkably linear —which implies good scalability with size— and holds for all indices, albeit at different slopes. The slope of a curve in this graph ("slope") is a measure of the hashing efficiency: the optimal slope ("opt") for each index is *1/sets*. The ratio of the slope to its optimal is a measure of its closeness to the optimal.

The most important observation here is that although the slopes of the curves are quite near the theoretical optimal slopes in each case, *small indices are closer to the optimal slopes than longer indices confirming increasing inefficiency with index length.*

To conclude, the choic of the index must strike a fine balance between the memory overhead to store a routing table and its associativity requirements. Both memory size and associativity negatively affect power consumption and performance of an actual IPStash device.

The above analysis pertains to information (memory overhead, required associativity) that we extract solely from routing tables. The rest of the paper deals with the analysis of architectural trade-offs in the context of designing a memory structure optimized for IP-lookup. This is the topic of Section V where we use the Cacti tool to study this problem.

TABLE I. Required memory for different indices (average values for 8 tables)

| INDEX | SETS | CLASS 3 | CLASS 2 | CLASS 1 | EXPANSION % | MEMORY OVHD | MEMORY OVHD (SKEWED) |
|---|---|---|---|---|---|---|---|
| 8 | 256 | 1,12,15[1] | 16,16,19 | 20,20,32 | 0.29 | 1.35 | 1.018 |
| 9 | 512 | 1,13,17 | 18,18,21 | 22,22,32 | 0.67 | 1.44 | 1.03 |
| 10 | 1K | 1,14,17 | 18,18,21 | 22,22,32 | 1.51 | 1.61 | 1.046 |
| 11 | 2K | 1,15,18 | 19,19,22 | 23,23,32 | 3.44 | 1.85 | 1.102 |
| 12 | 4K | 1,15,18 | 19,19,22 | 23,23,32 | 3.45 | 2.26 | 1.23 |
| 13 | 8K | 1,16,18 | 19,19,22 | 23,23,32 | 7.65 | 3.22 | 1.46 |
| 14 | 16K | 1,17,19 | 20,20,23 | 24,24,32 | 26.26 | 4.84 | 1.936 |
| 15 | 32K | 1,18,19 | 20,20,23 | 24,24,32 | 55.4 | 5.76 | 2.53 |
| 16 | 64K | 1,19,20 | 16,16,19 | 24,24,32 | 115.62 | 8.75 | 3.5 |

1. Classes are described by the tuple: (Lower bound, Index LSB, Upper bound)

TABLE II. Ultra-18 (SiberCore) power characteristics

| SEARCH RATE (MSPS) | ALL BLOCKS SEARCHED | | 1 BLOCK SEARCHED | |
|---|---|---|---|---|
| | POWER (WATT) | POWER PER Mb (WATT) | POWER (WATT) | POWER PER Mb (WATT) |
| 50 | 4.44 | 0.247 | 13.32 | 0.74 |
| 66 | 5.7 | 0.317 | 16.92 | 0.94 |
| 83 | 6.81 | 0.378 | 21.34 | 1.186 |
| 100 | 7.91 | 0.439 | 25.88 | 1.438 |

## IV. OTHER FEATURES OF THE ARCHITECTUES

### A Incremental Updates

According to [3] many network equipment design engineers share the view that it is not the increasing size of the routing tables but the super-linear increase in the number of updates that is going to hinder the development of next generation internet devices. The requirement for a fast update rate is essential for a router design. This is true because the routing tables are hardly static [6,10,14]. A real life worst-case scenario that routers are called to handle is the tremendous burst of BGP update packets that results from multiple downed links or routers. In such unstable conditions the next generation of forwarding engines requires bounded processing overhead for updates in the face of several thousand route updates per second.

Routing table update has been a serious problem in many TCAM-based proposals. The problem is that the more one optimizes the routing table for a TCAM the more difficult it is to modify it. Many times updating a routing table in a TCAM means inserting/deleting the route externally, re-processing the routing table, and re-loading it on the TCAM (a situation that stands for the trie based lookup schemes). In other proposals, there is provision for empty space distributed in the TCAM to accommodate a number of new routes before re-processing and re-loading the entire table is required [24]. This extra space, however, leads to fragmentation and reduces capacity. The updating problem becomes more difficult in "blocked" TCAMs where additional partitioning decisions have to be taken.

In contrast, route additions in IPStash are straightforward: a new route is expanded to the prefixes of the appropriate length if needed (no resorting is required), and it is inserted into the IPStash as any other prefix during the initial loading of the routing table. Deletions are also straightforward: the deleted route is presented to the IPStash to invalidate the matching entries having the same length as the deleted route.

### B Expanding the IPstash

As a result of CIDR, the trend for routing table sizes is a rapid increase over the last few years. It is hard to predict routing table sizes 5 —or, worse, 10— years hence. Thus, scaling is a required feature of the systems handling the Internet infrastructure, because they should be able to face new and partly unknown traffic demands.

IPStash can be easily expanded. There is no need for additional hardware and very little arbitration logic is required, in contrast to TCAMs which need at least a new priority encoder and additional connections to be added to an existing design. We consider this as one of the main advantages of our proposal. Adding in parallel more IPStash devices *increases associativity.* Length arbitration to select the longest match across multiple devices is now expanded outside the devices with a 32-bit wired-or arbitration bus which is a hierarchical extension of the length-arbitration bus discussed in Section II.A. Further details can be found in [8].

TABLE III. Cacti power and timing results for a 512k-entry IPStash device with 32 way associativity

| IPSTASH CONFIGURATION | | | ACCESS TIME (NS) | CYCLE TIME (NS) | MAX FREQ. (MHZ) | MAX THR. (MSPS) | POWER AT 100 MSPS (WATT) |
|---|---|---|---|---|---|---|---|
| INDEX BITS | BANKS | ASSOC | | | | | |
| 8 | 64 | 32 | 15.19 | 5.66 | 177 | 59 | — |
| 9 | 32 | 32 | 6.11 | 2.04 | 491 | 163 | 16.14 |
| 10 | 16 | 32 | 5.18 | 1.72 | 582 | 194 | 9.23 |
| 11 | 8 | 32 | 5.4 | 2.28 | 439 | 146 | 4.93 |
| 12 | 4 | 32 | 4.36 | 2.09 | 479 | 159 | 2.8 |
| 13 | 2 | 32 | 5.71 | 2.56 | 391 | 130 | 2.02 |
| 14 | 1 | 32 | 8.53 | 4.45 | 225 | 75 | — |

## V. DETAILED EXPLORATION OF THE DESIGN SPACE

We used Cacti 3.2 tool [28] to estimate performance and power consumption of IPStash. Cacti iterates over multiple cache configurations until it finds a configuration optimized for speed, power, and area. For a level comparison we examine IPStash and TCAMs at the same technology integration (0.15u).

To increase capacity in IPStash we add more associativity. This stems from the linear relation of routing table size and required associativity. We extended Cacti to handle more than 32-ways, but as of yet we are unable to validate these numbers. Thus, we use Cacti's ability to simulate multi-banked caches to increase size and associativity at the same time. In Cacti, multiple banks are accessed in parallel and are intended mainly as an alternative to multiple ports. We use them, however, to simulate higher capacity and associativity.

Our basis for comparison is the *Ultra-18* (18Mbit, 512K IPv4 entries) TCAM from *SiberCore* [25]. Ultra-18 is presently the top-of-line TCAM[1]. Table III shows the power characteristics of the Ultra-18. Since in our study we cannot scale IPStash arbitrarily (because of Cacti's powers-of-two restrictions) we chose to scale the TCAMs instead. Detailed characteristics presented in Table II allow us to project Ultra-18 power consumption for specific capacities. Our approach is to use IPStash memory overhead factors presented in Table I to scale TCAM capacity. For example, a 512K-entry IPStash with a 12-bit index has a memory overhead of 1.23 meaning that it can store a routing table of about 512/1.23 = 416K entries. Thus, we compare against a TCAM with same *scaled* capacity, i.e., a TCAM with 416K entries.

We use Cacti to study various configurations (adjusting associativity, number of sets, and number of banks) of a 512K-entry IPStash. An entry in our case contains the maximum number of prefix bits —aside from index bits— plus the corresponding mask (e.g., for a 12 bit index, 20+20 = 40 bits for tag), and data payload (8-bit port number). Table III shows power and latency results for some of the possible configurations where the associativity (of each bank) is fixed at 32. Power results are normalized for the same throughput —e.g., 100 *Million Searches Per Second* (*Msps*), a common performance target for many TCAMs. We restrict solutions to those with a memory overhead less than 2 (Table I). The reasoning is that TCAMs also have a hidden memory overhead to support wildcards which is exactly 2.

[1] Recently (Feb.-2004) Netlogic Microsystems released a new TCAM using 0.13u process technology.

Two more changes are needed in Cacti to simulate IPStash. The first is the extra wired-or bus required for length arbitration. The arbitration bus adds both latency and power to each access. Using Cacti's estimates we compute the overhead to be less than 0.4 Watts (at 100 Msps). Our estimates for the arbitration bus are based on the power and latency of the cache's bitlines. We consider length arbitration as a separate pipeline stage in IPStash which, however, does not affect cycle time —address decoders define cycle time in all cases. The second change concerns the support for skewed associativity. Skewed index construction (rotations and XORs) introduces negligible latency and power consumption to the design. However, a skewed-associative IPStash requires separate decoders for the wordlines — something Cacti does not do on its own. We compute latency and power overhead of the separate decoders in all cases. We conclude that the skewed-associative IPStash is slightly *faster* than a standard IPStash while consuming about the same power. The reason is that the decoders required in the skewed-associative case are faster than the monolithic decoder employed in the standard case. At the same time although each of the small decoders consumes less power than the original monolithic decoder, all of them together consume slightly more power.

With our modifications, Cacti shows that a 512K-entry, 32-way, IPStash easily exceeds 100 Msps. In any configuration, pipeline cycle time is on the order of 2 to 5 ns. Power consumption at 100 Msps starts at 2.13 W (including length arbitration and skewing overhead) with a 13-bit index and increases with decreasing index. In the extreme case of an 8-bit index, power is overwhelming mainly due to routing overhead (among banks). Power results are normalized for the same throughput (100 Msps) instead of frequency. Thus, the operational frequency of IPStash may not be the same as in TCAMs —it is in fact higher. Results are analogous for the 200 Msps level performance.

Results for the 32-way IPStash configurations show a clear trade-off between power and performance. In the next section we introduce a power management technique for IPStash and present results for the most appealing configurations in terms of power *or* performance in the entire design space of IPStash devices.

### A Power Management in IPStash

As we have shown in the previous section, for the same performance, IPStash power consumption is significantly lower than the announced minimum power consumption of the Ultra-18 with optimal power management. Power management in the TCAM typically requires both optimal partitioning of the routing tables and external hardware to selectively power-up individual TCAM blocks.

In this section we introduce a novel power management technique for IPStash that is simple, transparent, and often very effective. The concept is to assign *favorite* —but not necessarily *exclusive*— associative ways or banks of ways to different prefix classes. In the following we refer to banks of ways but our discussion applies equally well to individual associative ways. The hope is that, for the most part, different classes end up occupying different banks. Since in our LPM we search classes consecutively, when a class occupies specific banks we restrict our search solely to those.

This power management technique can be implemented with very little hardware. First, we assign favorite banks to sets of classes in a very simple manner: Class 1 (the largest) favors the leftmost banks while the combination of Class 2 and Class 3 favors the rightmost banks. All the classes intermix somewhere the middle. "Bank-favoritism" is exhibited on prefix insertion only: we simply steer Class-1 prefixes to the left and Class-2 and 3 prefixes to the right. For each bank two bits describe three possibilities for its contents: i) contains Class-1 prefixes only, ii) contains Class-2 and Class-3 only, iii) contains all three classes. Depending on the class we are searching in our LPM, only the relevant banks participate in the access and search.

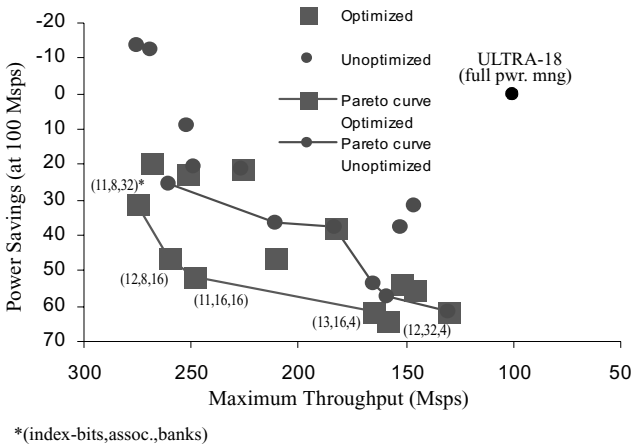


Fig. 12. Power vs. Speed for optimized (power managed) and un-optimized IPStash compared to a state-of-the-art TCAM. In each case, *Pareto* curves denote the best options in the design space.

Cacti incorporates a simple model to simulate multi-bank caches which is applicable in our case. Cacti considers each bank as fully independent: every bank has its own independent address and data lines. Cacti includes a routing overhead that represents power and time penalty for driving address and data lines to each bank.

Assuming a 512k-entry IPStash with 16 banks each consisting of 16 ways, our simulations show that 84% of the total associativity is devoted to pure set-associative ways (57% associativity for Class-1 prefixes, 27% for the Class-2 and Class-3 prefixes) and 16% of the associativity is devoted to mixed classes. This means that upon arrival of an incoming packet, in the first lookup (Class-1) only 73% of the banks (12 banks) need to be searched and only 42% of the banks (7 banks) are needed for the other two sequential searches. Average power consumption in this case is reduced by 37.8%.

Fig. 12 presents results for all possible configurations (1 to 64 banks, 4 to 32 associativity per bank) of a 512K IPStash with indices of 11-14 bits. The horizontal dimension represents the maximum search rate (in Msps) that a specific IPStash can achieve and the vertical dimension represents maximum power *reduction* compared to the *scaled* power consumption of the ULTRA-18 TCAM with full memory management. All power results are normalized for the same throughput —100 Msps.

IPStash power consumption without any power management is 61% lower compared to the fully-power-managed ULTRA-18. When we employ power management in IPStash, a further improvement in power consumption is achieved. In our case, power management introduces negligible overhead, needing no additional external hardware or effort. Considering the search throughput, IPStash devices easily exceed the current top-of-the-line performance of 100 Msps. In some configurations more than 250 Msps are achieved.

### B Effects of Packet Traffic on Power and Latency

As we have discussed, the concept for longest prefix match in IPStash is to iteratively search prefix classes —usually three in our study— for progressively shorter prefixes until a match is found. For the analysis in Section V we assume worst case

behavior, that is, *all classes are always searched regardless of where the first hit occurs*.

In reality, we can stop the search on the first (longest) match. As more incoming IP addresses hit, for example, on Class 3 prefixes (the first class searched), fewer memory accesses per search are required, thus both average search latency and power consumption are reduced. An optimized IPStash device should operate in this fashion. The distribution of hits to classes for a specific traffic trace determines the benefits in power and latency. Assuming a uniform distribution of hits to three classes we can reduce power and latency by a factor of 1/3.

Although many organizations, make packet traces publicly available trough the National Laboratory for Applied Network Research (NLANR) [20], privacy considerations dictate the anonymization of IP addresses. Unfortunately, this prevents us from obtaining reliable hit-distribution results when we use anonymized traffic with non-anonymized routing tables. We note here, that the hit distribution for some expected traffic can drive the initial class selection. It might be beneficial to opt for a sub-optimal class selection (in terms of memory-overhead and required associativity) which, however, optimizes the average number of accesses per search.

## VI. RELEATED WORK

TCAMs offer good functionality, but are expensive, power hungry, and less dense than conventional SRAMs. In addition, one needs to sort routes to guarantee correct longest prefix match. This often is a time and power consuming process in itself. Two solutions for the problem of updating/sorting TCAM routing tables have been recently proposed [9,24]. The problem of power consumption in TCAM-based routers attracts significant attention by researchers. Liu [12] uses a combination of pruning techniques and logic minimization algorithms to reduce the size of TCAM-based routing tables. However, power consumption still remains quite high. Zane, Narlikar and Basu [29] take advantage of the effort of several TCAM vendors to reduce power consumption by providing mechanisms to enable and search only a part of a TCAM much smaller than the entire TCAM array. The authors propose a bit-selection architecture and partitioning technique to design a power-efficient TCAM architecture. In [18], the authors propose to place TCAMs on separate buses for parallel accesses and introduce a paged-TCAM architecture to increase throughput and reduce power consumption. The idea of a "paging" TCAM architecture is further explored in [21,30] in order to achieve new levels of power reduction and throughput. Our proposal is similar in spirit but distinctly different in implementation since we advocate separation of storage (in an SRAM set-associative memory array) and search functionality (variable tag match and length arbitration). We believe that this separation results in the most efficient implementations of the "blocking" or paging concept. Furthermore, our effort is centered in fitting a routing table in the most efficient manner in the least associative array possible.

Many researchers employ caches to speed up the translation of the destination addresses to output port numbers [1,2,4,13,27]. Studies for Internet traffic [19] show that there is a significant locality in the packet streams that caching could be a simple and powerful technique to address per-packet processing overhead in routers. Most software-based routing table lookup algorithms optimize the usage of cache in general purpose processors, such as algorithms proposed in [4,19].

Our approach is different from all previous work. Instead of using a cache in combination with a general-purpose processor or an ASIC routing machine, we use a stand-alone set-associative architecture. IPStash offers unparalleled simplicity compared to all previous proposals while being fast and power-efficient at the same time.

## VII. CONCLUSIONS

In this paper, we propose a set-associative architecture called IPStash which abandons the TCAMs in IP-lookup applications. IPStash overcomes many problems faced by TCAM designs such as the complexity needed to manage the routing table, power consumption, density and cost. IPStash can be faster than TCAMs and more power efficient while still maintaining the simplicity of a content addressable memory.

The recent turn of the TCAM vendors to power-efficient blocked architectures where the TCAM is divided up in independent blocks that can be addressed externally justifies our approach. Blocked TCAMs resemble set-associative memories, and our own proposal in particular, but their blocks are too few, their associativity is too high, and their comparators are embedded in the storage array instead of being separate. In our mind, we see no reason to use a fully-associative, ternary, content-addressable memory to do the work of a set-associative memory.

What we show in this paper is that associativity is a function of the routing table size and therefore need not be inordinately high as in blocked TCAMs with respect to the current storage capacities of such devices. What we propose is to go all the way, and instead of having a blocked fully-associative architecture that inherits the deficiencies of the TCAMs, start with a clean set-associative design and implement IP-lookup on it. We show how longest prefix match can be implemented by iteratively searching *classes* of (increasingly) shorter prefixes. Prefix classes allow us to hash the routing table multiple times (each time using an optimized index) for insertion in IPStash. Multiple-hashing coupled with skewed associativity results in a required associativity for routing tables impressively close to optimal.

Using Cacti, we study IPStash using 8 routing table sizes and find that it can be more than twice as fast as the top-of-the-line TCAMs while offering up to 64% power savings (for the same throughput) over the announced minimum power consumption of commercial products. In addition, IPStash exceeds 250 Msps while the state-of-the-art performance for TCAMs (in the same technology) currently only reaches about 100 Msps.

We believe that IPStash is the natural evolutionary step for large-scale IP-lookup from TCAMs to associative memories. We are working on expanding IPStash to support many other networking applications such as IPv6, NAT, MPLS, the handling of millions of "flows" (point-to-point Internet connections) by using similar techniques as in IP-lookup.

## ACKNOWLEDGEMENTS

This work is supported by Intel Research Equipment Grant #15842.

## REFERENCES

[1] J. Baer, D. Low, P. Crowley, and N Sidhwaney. Memory Hierarchy Design for a Multiprocessor Lookup Engine. Pact 2003, Septemper 2003.

[2] G. Cheung and S. McCanne, "Optimal Routing Table Design for IP Address Lookups Under Memory Constraints." *IEEE INFOCOM*, pp. 1437-44, 1999.

[3] E. Chang, B. Lu and F. Markhovsky, "RLDRAMs vs. CAMs/SRAMs", Part 1 and 2, in CommsDesign, 2003.

[4] T. Chiueh and P. Pradhan, "Cache Memory Design for Network Proces-

sors." *Proc. High Performance Computer Architecture*, pp. 409-418, 1999.
[5] A. Gallo, "Meeting Traffic Demands with Next-Generation Internet Infrastructure." *Lightwave*, 18(5):118-123, May 2001.
[6] G. Huston, "Analyzing the Internet's BGP Routing Table." *The Internet Protocol Journal*, 4, 2001.
[7] IDT. http://www.idt.com
[8] S. Kaxiras and G. Keramidas, "IPStash: A Power Efficient Memory Architecture for IP lookup", In Proc. of MICRO-36, November 2003.
[9] M. Kobayashi, T. Murase, A. Kuriyama, "A Longest Prefix Match Search Engine for Multi-Gigabit IP Processing." In *Proceedings of the International Conference on Communications (ICC 2000)*, pp. 1360-1364, 2000.
[10] C. Labovitz, G.R. Malan, F. Jahanian, "Internet Routing Instability." The *IEEE/ACM Transactions on Networking*, Vol. 6, no. 5, pp. 515-528, 1999.
[11] B. Lampson, V. Srinivasan, G. Varghese, "IP-lookups Using Multiway and Multicolumn Search." Proceedings of *IEEE INFOCOM*, vol. 3, pages 1248-56, April 1998.
[12] H. Liu, "Routing Table Compaction in Ternary CAM." *IEEE Micro*, 22(1):58-64, January-February 2002.
[13] H. Liu, "Routing Prefix Caching in Network Processor Design." *IEEE ICCCN2001*, October 2001.
[14] R. Mahajan, D. Wetherall, T. Anderson, "Understanding BGP Misconfiguration." *SIGCOMM '02*, August 2002.
[15] Micron Technology. http://www.micron.com
[16] Netlogic microsystems. http:// www.netlogicmicro.com
[17] S. Nilsson and G. Karlsson, "IP-address lookup using LC-tries." *IEEE Journal of Selected Areas in Communications*, vol. 17, no. 6, pages 1083-92, June 1999.
[18] R. Panigraphy and S. Sharma, "Reducing TCAM Power Consumption and Increasing Thoughput", *Proc. of HotI'02,* Stanford, California, 2002.
[19] C. Partridge, "Locality and Route Caches." *NSF Workshop on Internet Statistics Measurement and Analysis*, 1996.
[20] Passive Measurement and Analysis project, National Laboratory for Applied Network Research. http://pma.nlanr.net/PMA
[21] V. C. Ravikumar, R. Mahapatra and L. Bhuyan, "EaseCAM: An Energy and Storage Efficient TCAM-based Router Architecture", *IEEE Micro*, 2004.
[22] RIPE Network Coordination Centre. http://www.ripe.net
[23] A. Seznec, "A case for two-way skewed-associative cache," Proceedings of the 20th *International Symposium on Computer Architecture*, May 1993.
[24] D. Shah and P. Gupta, "Fast Updating Algorithms for TCAMs." *IEEE Micro*, 21(1):36-47, January-February 2001.
[25] Sibercore Technology. http://www.sibercore.com
[26] V. Srinivasan and G. Varghese, "Fast Address Lookups Using Controlled Prefix Expansion." *ACM Transactions on Computer Systems*, 17(1):1-40, February 1999.
[27] B. Talbot, T. Sherwood, B. Lin, "IP Caching for Terabit Speed Routers." *Global Communications Conference*, pp. 1565-1569, December, 1999.
[28] S.J.E. Wilton and N.P. Jouppi, "Cacti: An Enhanced Cache Access and Cycle Time Model." *IEEE Journal of Solid-State Circuits*, May 1996.
[29] F. Zane, G. Narlikar, A. Basu, "CoolCAMs:Power-Efficient TCAMs for Forwarding Engines."*IEEE INFOCOM*,April 2003.
[30] K. Zheng, C. Hu, H. Lu and B. Liu, "An Ultra High Thoughput and Power Efficinent TCAM-Based IP Lookup Engine", *IEEE Infocom,* 2004.