IPStash: a Power-Efficient Memory Architecture for 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

High-speed routers often use commodity, fully-associative, TCAMs (Ternary Content Addressable Memories) to perform packet classification and routing (IP-lookup). We propose a memory architecture called IPStash to act as a TCAM replacement, offering at the same time, better functionality, 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. We propose a memory architecture similar to set-associative caches but enhanced with mechanisms to facilitate IP-lookup and in particular longest prefix match. *To perform longest prefix match efficiently in a set-associative array we restrict routing table prefixes to a small number of lengths using a controlled prefix expansion technique. Since this inflates the routing tables, we use skewed associativity to increase the effective capacity of our devices. Compared to previous proposals, IPStash does not require any complicated routing table transformations but more importantly, it makes incremental updates to the routing tables effortless. The proposed architecture is also easily expandable. Our simulations show that IPStash is both fast and power efficient compared to TCAMs. Specifically, IPStash devices —built in the same technology as TCAMs— can run at speeds in excess of 600 MHz, offer more than twice the search throughput (>200Msps), and consume up to 35% less power (for the same throughput) than the best commercially available TCAMs when tested with real routing tables and IP traffic.*

1. Introduction

A critical function in network routers is packet classification—in other words, determining routing and traffic 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) [34].

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. Today's leading (2.5, 5, and 10 Gbit/sec) network processors such as Intel's IXP 2850 [17], EZChip's NP-1 [12], Agere's APP550 [30], IBM's PowerNP (NP4GS3) [15] and Vitesse's IQ2200 [42] achieve the necessary lookup rate using a combination of high speed memories and specialized access hardware. Another direction concentrates on partitioning routing tables in optimized data structures, often in tries (a form of trees), so as to reduce as much as possible the average number of accesses needed to perform LPM [39,31,6,8]. Each lookup however, requires several *dependent* (serialized) memory accesses stressing conventional memory architectures to the limit. Memory latency and not bandwidth is the limiting factor with these approaches.

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 —routes ending with wildcards— Ternary CAMs (TCAMs) are used. TCAMs have an additional "don't care" bit for every tag bit. When the "don't care" bit is set the tag bit becomes a wildcard and matches anything. The ternary capability of TCAMs makes them an attractive solution for the IP-lookup problem and thus have found acceptance in many commercial products. Several companies (IDT [16], Netlogic [29], Micron [28], Sibercore [38]) 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 [37,20], 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 hamper the wide deployment of 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 [16,38]. 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 [13]. 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 [43,16,29]. 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*. 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:

1. We propose a set-associative memory architecture for IP-lookup and we show how we can solve the problem of Longest Prefix Match in IPStash. Since we do not support "don't care" bits, we allow only a limited number of different prefix lengths. This inflates the routing tables but still requires about the same number of bits as TCAM.
2. Because of the increased size of the routing tables in the IPStash, we show how *skewed associativity* can be applied with great success to increase effective capacity.
3. We use real data to validate our assumptions with simulations. We use the Cacti tool to estimate power consumption and we show that IPStash consumes up to 35% less power than the best commercial available blocked TCAMs. It is also possible to optimize the power consumption of IPStash (for example by selectively powering-down ways that contain irrelevant entries) but we have not expanded into such techniques in this paper.

Structure of this paper—Section 2 presents the IPStash architecture and our implementation of the LPM algorithm. In Section 3 we analyze a technique (skewed associativity) to increase the effective capacity of our device while in Section 4 we show that IP-lookup needs associativity depending on the routing table size. In Section 5 we discuss some special cases and the expandability of our proposal. Section 6 provides simulation results for power consumption. Finally, Section 7 presents related work and Section 8 offers our conclusions.

2. 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.*

IEEE COMPUTER SOCIETY

2.1. Longest Prefix Match in IPStash

The concept for Longest Prefix Match in IPStash is to iteratively search the set-associative array for progressively shorter route prefixes until a match is found. Let us consider for the moment only a limited set of prefix lengths, for example 24-bit, 20-bit, and 16-bit prefixes but no other length. For a routing table consisting solely of such prefixes and for a 32-way, 4096-set (12-bit index) IPStash, the operation (shown in Figure 1) is as follows: we insert the longest, 24-bit prefixes using their rightmost (LSB) 12 bits as index and their leftmost (MSB) 12 bits as tag. We also store the prefix length with the tag. Similarly, we insert the 20-bit prefixes using their rightmost 12 bits as index and leftmost 8 bits as tag, and the 16-bit prefixes using their rightmost 12 bits as index and leftmost 4 bits as tag.

If we fit all of these route prefixes in IPStash without any conflicts —*an important goal in this work*— we search for the longest prefix match to an IP address in three steps, as shown in Figure 1:

1. Step 1: We first try to match a 24-bit prefix. To index for a 24-bit prefix we must use the same index bits used to insert 24-bit prefixes in IPStash. Thus, bits 12:23 of the IP address form the index. We read the indexed set and this gives us 32 possible results. Among these results we exclusively look for a 24-bit prefix —the length of the prefix is kept with the tag— whose tag matches bits 0:11 of the address. If we find such a match then we have found the longest prefix match.
2. Step 2: Otherwise, we use bits 8:19 of the IP address as index, now hoping to find a 20-bit prefix to match. Again, we access the appropriate set and we search the 32 possible results but now for a length-20 match and for an 8-bit tag match.
3. Step 3: Similarly, for the last case we try to match a 16-bit prefix using bits 4:15 of the IP address as index and checking for a 4-bit tag.

Because a hit in IPStash requires two conditions to be true (tag match and length match) the final stage of the set-associative structure is more complex than that of a vanilla set-associative cache. We account for this in later sections.

2.2. Fitting a real routing table in IPStash

One can imagine that we can extend this to more than three prefix lengths to include all possible lengths but this would be 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 expand prefixes of different lengths to a small set of fixed lengths. The choice of which prefixes to expand and how much depends largely on the prefix length distributions of the routing tables.

Many researchers have observed a distinct commonality in the distribution of prefix lengths in routing tables [39,31,6,3] 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 [14]. Figure 2 shows the distribution of prefix lengths for three tables taken from [4] (log-scale view on top for clarity and normal view below). We can easily draw some general conclusions —also noted by other researchers— from the graphs in Figure 2: 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 of the prefixes have lengths between 16 and 24 bits.

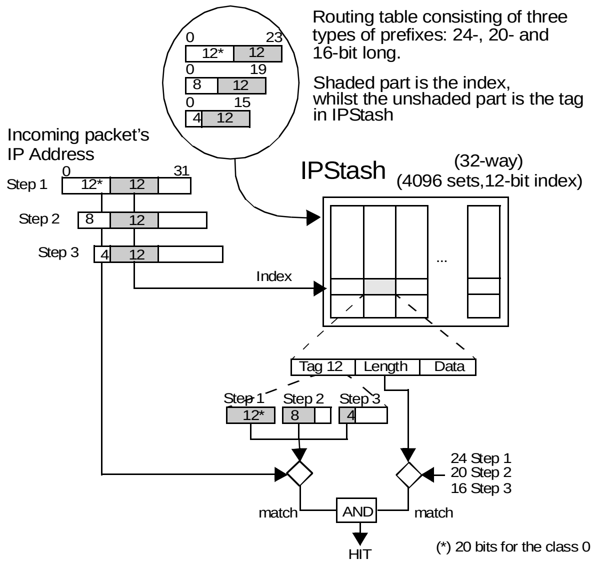
Routing table consisting of three types of prefixes: 24-, 20- and 16-bit long.
Shaded part is the index, whilst the unshaded part is the tag in IPStash
0
23
12*
12
19
8
15
4
Incoming packet's IP Address
31
Step 1
Step 2
Step 3
IPStash
(32-way)
(4096 sets,12-bit index)
Index
...
Tag 12
Length
Data
24 Step 1
20 Step 2
16 Step 3
match
AND
HIT
(*) 20 bits for the class 0

Figure 1. IPStash operation for three prefix lengths

These observations lead to a natural categorization of the routing table into three major classes and two shadow classes:

- **Class 1** contains all the prefixes from 21 to 24 bits. 21-bit, 22-bit, and 23-bit prefixes are all expanded to 24 bits. The 21-bit prefixes are expanded 8-fold, the 22-bit ones 4-fold, and the 23-bit 2-fold. In addition, in this class we add a "shadow class," Class 0 —explained below— for the prefixes with more than 24 significant bits.
- **Class 2** contains all the prefixes from 17 to 20 bits. 17-bit, 18-bit, and 19-bit routes are all expanded to 20 bits similarly to above.
- **Class 3** contains all the prefixes from 8 to 16 bits. In this class we can afford to expand the 8-bit prefixes 256-fold (9-bit 128-fold, 10-bit 64-fold, and so on) because there are so few of them.
- **Shadow Class 4** contains all the prefixes from 1 to 7 bits and it is *empty.* No prefixes shorter than 8 bits appear in BGP tables although CIDR does not preclude such possibility. Because it is unlikely to encounter such prefixes all relevant work ignores them. However, IPStash could handle Class 4 prefixes expanded to 7 bits (up to 128 entries in total).
- **Shadow Class 0** contains all prefixes longer than 25 bits. The reason this is a special shadow class is because we fold it on top of Class 1. We use exactly the same index as in Class 1 but the tag is expanded

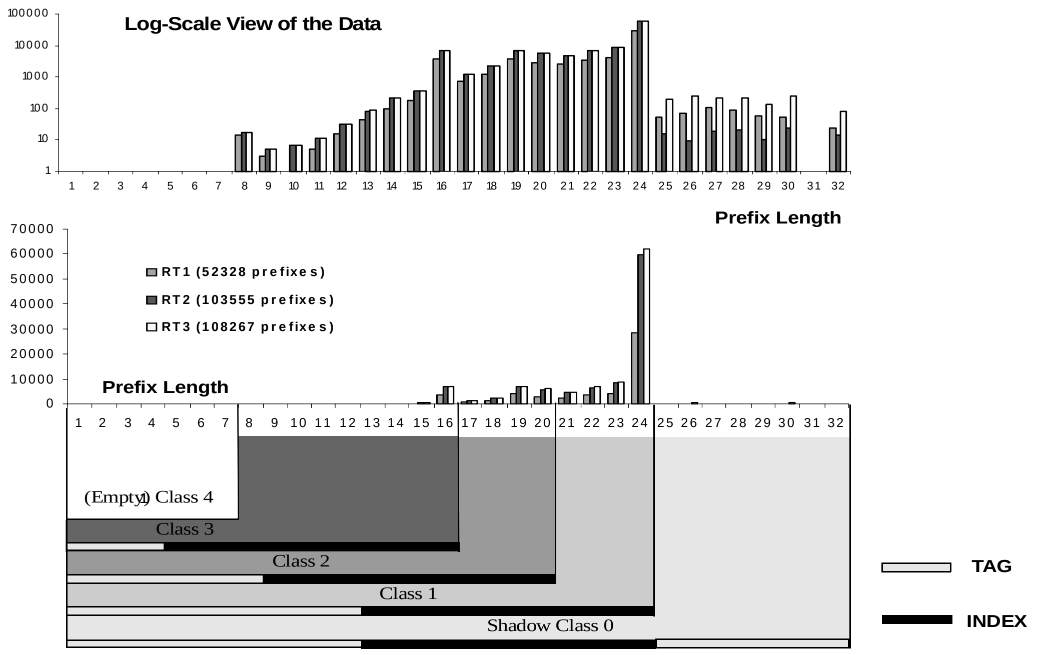

Figure 2. Distribution of prefix lengths for 3 routing tables

to 20 bits. The tag contains the 12 tag bits of Class 1 plus any additional prefix bits beyond 24. Class 0 entries are matched with a Class 1 access but the expanded tag disambiguates among them. The part of the tag that is matched depends on the unexpanded length stored along with the tag. For this class only, no prefix expansion is required because its index always precedes the wildcard bits. Class 0 is exceedingly small compared to the very large Class 1, thus folding Class 0 entries on top of Class 1 entries has negligible effects.

Prefix expansion necessitates a more complex length match to determine a hit: within a class, IPStash must select the longest *unexpanded-length* entry (the *unexpanded* prefix length is stored with the tag).

Prefix expansion can happen externally or internally in IPStash. External prefix expansion requires the cooperation of an entity using the IPStash (e.g., a network processor) so that only prefixes of the correct lengths are stored. Internal expansion is straightforward requiring only a small FSM and counters to fill the wildcard bits, but it makes prefix insertion in IPStash a variable-time operation as seen from the outside. Either solution is acceptable, but for simplicity we only consider the former in this paper.

The effect of expanding routes in three classes is to inflate the routing tables. The effects of this inflation are shown in Table 1 for the three routing tables RT1, RT2 and RT3 of [4]. The routing tables almost double in size with the route expansion. In general, this means that IPStash capacity should be about twice the size of routing table we intent to store. This, however, is not excessive overhead compared to TCAMs. The true capacity of a TCAM is twice its nominal capacity because of the "don't care" bits —for every storage bit there is a corresponding "don't care" bit, plus additional comparator hardware. In this work, we compare devices of approximately equal actual storage in bits (including hidden bits). Thus, we compare a 128K-entry IPStash to a 64K-entry *(nominal capacity)* TCAM. Moreover, IPStash is composed of SRAM arrays which are about a factor of 2 denser than current TCAM technology.

Since we would use a 64K-entry TCAM for RT1 and 128K-entry TCAMs for RT2 and RT3 we use 128K-entry and 256K-entry IPStash devices respectively. IPStash stores the three routing tables with considerable but not complete success. Table 2 shows the IPStash configurations used for each of the routing tables and the resulting conflicts in the set-associative arrays. Conflicts correspond to a very small portion of the routing table.

Conflicts cannot be accommodated in IPStash since it is not a cache but the *main storage* for the routing table. In Section 3 we concentrate our efforts on increasing the effective capacity of IPStash devices.

	INITIAL ROUTES	EXPANDED ROUTES	INCREASE	DATE
RT1	52328	102550	1.96	Nov 12, 1999
RT2	103555	194541	1.88	Oct. 1, 2001
RT3	108267	202094	1.87	Oct 1, 2001

Table 1: The effect of the route expansion

	EXPANDED ROUTES	IPSTASH CONFIGURATION	UNRESOLVED CONFLICTS
RT1	102550	128K ENTRIES, 32 ASSOC.	1685 (~3.2%)
RT2	194541	256K ENTRIES, 64 ASSOC.	566 (~0.55%)
RT3	202094	256K ENTRIES, 64 ASSOC.	979 (~0.9%)

Table 2: Resulting conflicts for the three tables

2.3. Sensitivity Analysis for Classes

The choice of classes is a trade-off between mem-

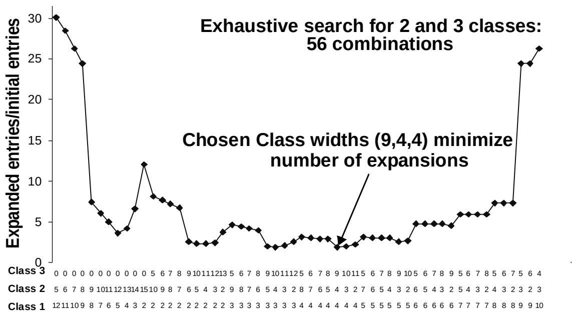

Figure 3. Sensitivity analysis for 3 classes

ory requirements of the expanded routing table and the maximum number of accesses required for our iterative LPM. Few classes means few accesses per lookup but high memory requirements, and vice-versa. These are opposing trends with respect to both power consumption and lookup latency.

To understand the effect of Class choice on memory requirements we examined dividing the 8- to 24-bit prefixes into two, three and four main classes (excluding the two shadow Classes 0 and 4 which always remain the same). With only two main classes, almost any choice of class boundaries leads to exorbitant memory requirements. With four classes we raise the maximum number of accesses per lookup to 4. Figure 2 shows the effects of class boundaries for two and three classes. The horizontal axis shows the class widths while the vertical axis shows the memory requirements normalized to the original routing table. A casual study of the distribution of prefix lengths in Section 2.2 led us to one of the best configurations —with minimal memory requirements— for three classes (pointed out on the graph).

Because the effect of class boundaries depends on the actual prefix length distribution, we envision IPStash as a configurable device where the classes are set during power-up. For the sake of simplicity in the rest of this paper we restrict our examples to the aforementioned classes of Section 2.2.

3. Increasing effective capacity

3.1. Skewed associativity

Barring a change in IPStash geometry (associativity vs. number of sets), a change in the total capacity of an IPStash device, or additional hardware structures, our main proposal to increase effective capacity is based on Seznec's idea of a skewed associativity [36]. 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 (32 or 64 in our case). 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. This has the property of reducing the overall number of conflicts.

One way to think about skewed associativity is to

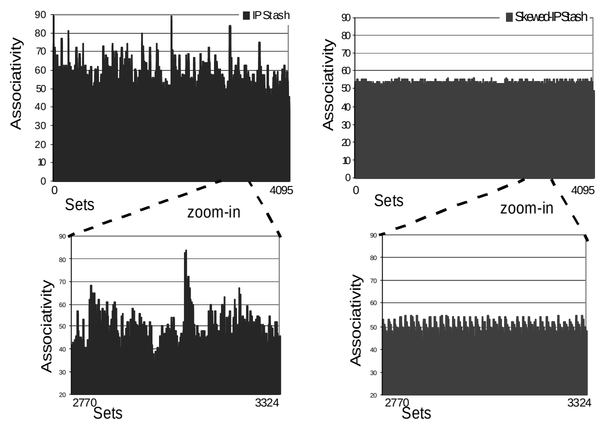

Figure 4. Original IPStash and skewed IPStash comparison (for RT3)

view it as an 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 Figure 4 shows how RT3 is loaded into an "unlimited-associativity" IPStash —without restriction to the number of ways. The horizontal dimension represents the sets (4096) and the vertical dimension the set-associative ways. As it is depicted in the graph, RT3 needs anywhere from 23 to 89 ways. If RT3 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 Figure 4) is to smooth-out the jagged edges of the original graph —in some sense to increase the entropy (disorder) of the system.

We produced the skewed-associative graph in Figure 4 using a simple skewing technique. The key issue in IPStash is that we do not have so much freedom to create 32 (or 64) distinct skewing functions because we have only few significant bits to exploit. Our compromising solution is to create only 8 (4 for Class-3) different skewed indices instead of 32 or 64 as the hardware associativity would call for. Each index is used on a bank of 4 ways in the 32-way IPStash —8 in the 64-way IPStash. Although this technique might not give us optimal results it has the desirable characteristic of curbing the increase in power consumption because of the multiple distinct decoders. Skewed indices are created as shown in Figure 5:

- For Class 1,0, and 2: The rightmost 8 bits of the original index are XORed with the 8 rightmost bits of the tag, right-rotated once for each different skewed index (for a total of 8 times). The leftmost 4 bits of the original index form the beginning of the skewed index.
- For Class 3: Because the tag in this case is only 4-bits wide we XOR it with the rightmost 4 bits of the original index, right-rotating it once for each skewed index. The leftmost 8 bits now form the left part (beginning) of the skewed index. Here, the 8 rotations only result in 4 distinct results, each used in two different banks.

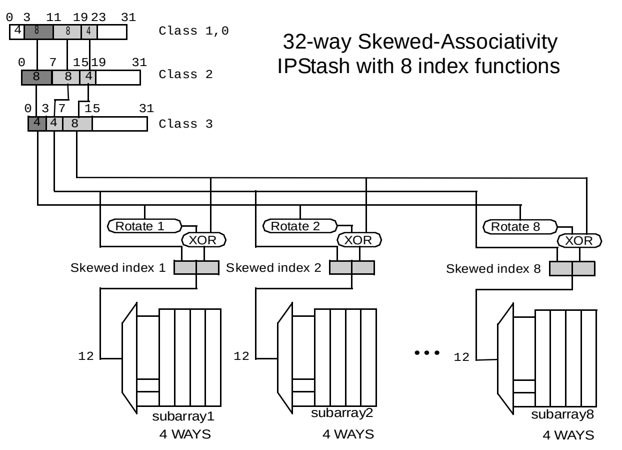

Figure 5. Skewed indices

	RT1 on 32-way		RT2 on 64-way		RT3 on 64-way	
Initial Routes	52328		103555		108267	
Expanded Routes	102550		194541		202094	
	STD	SKW	STD	SKW	STD	SKW
Total Conflicts	1685	0	566	0	979	0
Min associativity	9	21	22	40	23	42
Max associativity	49	30	86	54	89	56
Average associativity	25	25	47.5	47.5	49.3	49.3
Standard Deviation (σ)	11.52	1.65	15.64	2.64	16.18	2.69

Table 3: Skewed IPStash Detailed Comparison

Std Deviation range (around mean)	Confidence Interval (probability of falling inside range)	Required Associativity					
		RT1		RT2		RT3	
		Std	Skd	Std	Skd	Std	Skd
σ(±0.5σ)	0.6836	31	26	56	49	58	51
2σ (±σ)	0.9543	37	27	64	51	66	53
3σ (±1.5σ)	0.9973	43	28	71	52	74	54
4σ (±2σ)	0.9999	49	29	79	53	82	55

Table 4: Probabilistic associativity requirements

Variations of this skewing also worked very well or better and other techniques might prove even more successful. For the purposes of this work the skewing in Figure 5 is sufficient to illustrate our case. Skewing incurs a small hardware cost since the mapping functions are chosen for their simplicity.

Table 3 quantifies the effects of applying skewed associativity to IPStash devices. Without skewed-associativity Table 3 shows that a 32-way IPStash would have 1658 conflicts for RT1 (similarly for RT2 and RT3). This is because as RT1 is loaded into the IPStash each set is filled differently. Some sets receive only 9 entries which is the *Min associativity* required by RT1 while some sets receive 49 distinct entries which is the *Max associativity*. The average number of entries is the *Average associativity*. While —not surprisingly— the average associativity of the skewed-associative and the standard IPStash is the same for all cases, the *Standard Deviation* (σ) from the mean is what distinguishes them and makes all the difference for conflicts. This is a confirmation of the graphical results shown in Figure 4. In our examples the *Max associativity* for the skewed-associative cases does not exceed the predefined associativity (32 or 64) of the IPStash devices in any case.

3.2. Memory bounds

As we have shown, skewed associativity successfully reduces conflicts to zero. Results presented in this section also give us a rudimentary tool to estimate the capacity of IPStash devices with respect to routing tables.

Using standard deviation we can compute the required associativity for a *Confidence Interval (CI)*, i.e., the probability that the routing table fits in IPStash. Table 4 shows the results for a CI up to a 0.9999. This method is a convenient tool to check IPStash requirements of individual routing tables.

4. Associativity and routing table size

The initial premise of our work was that IP-lookup only needs associativity depending on the routing table size; *not full associativity*. Given a skewed-associative IPStash using a 12-bit index (4096 sets), Figure 6 shows the relationship of the required associativity to the original unexpanded size for our three routing tables (RT1, RT2 and RT3) and the MAE-West routing tables (MW1, MW2 and MW3) used in the traffic simulations of Section 6.2. This relationship is remarkably linear and it holds for non-skewed associative architectures and for other indices as well, albeit at different slopes. There are two important observations here:

- The slope of the curve is 0.0005, while the optimal slope is one over the number of sets (1/4096 = 0.00024) if one considers that the expanded tables are about twice the original size then the slope becomes 2/4096 = 0.00048. This means that our design is nearly optimal with respect to the expanded routing tables. In contrast, the slope for the fully-associative case is 1.
- The slope of the curve is constant which implies that our design scales well.

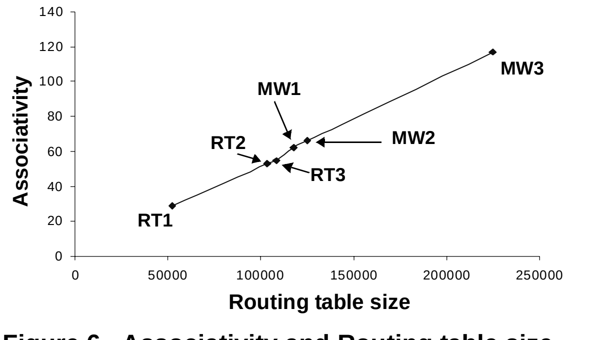

Figure 6. Associativity and Routing table size

5. Other features of the architecture

5.1. Incremental Updates

The requirement for a fast update rate is essential for a router design. This is true because the routing tables are hardly static [21,14]. Many changes in the routing tables occur due to changes in network topology. In addition, reboots of neighboring routers or BGP misconfigurations [26] do appear to occur every few days in real-life traces. 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 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. 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 [37]. 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 which are then inserted into the IPStash as any other prefix during the initial loading of the routing table.

Deletions are also straightforward: the deleted route is expanded into prefixes of the appropriate class length. The expanded prefixes are then presented to the IPStash to invalidate the matching entries having the same unexpanded length as the deleted route.

5.2. Expanding the IPstash

As a result of CIDR, the trend for routing table sizes is a rapid increase over the last few years [13,14]. 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.* Figure 7 shows the proposed scheme which actually resembles a cache-coherent bus. All IPStash devices are on the same (logical) buses. They accept requests (incoming IP addresses plus a command) from the request bus and reply with the output port information when they have a hit on the result bus. In case of multiple hits in different devices a 32-bit arbitration bus is used. All three logical busses can be multiplexed on a single physical 40-bit bus (32-bits for arbitration, IP addresses, or prefixes, plus 5 bits prefix length, plus 3 bits command).

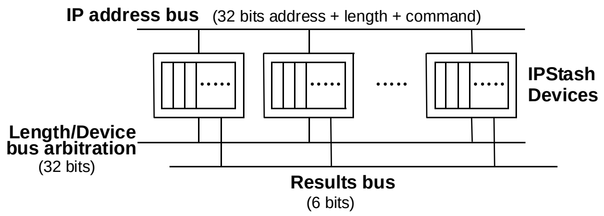
IP address bus (32 bits address + length + command)
IPStash Devices
Length/Device bus arbitration (32 bits)
Results bus (6 bits)

Figure 7. Multiple IPStash devices

Arbitration works as follows: If a device has a hit before any other device it is clearly the winner because it found the longest prefix (further search is inhibited in the rest of the devices). When multiple devices simultaneously have a hit, they output the original unexpanded length of their hit on the arbitration bus by asserting the wire that corresponds to their length. Every device sees each other’s length and a self-proclaimed winner outputs its result on the bus in the next cycle. All other devices whose length is not the largest on the arbitration bus keep quiet. This synchronous scheme works if all the classes are searched in lockstep so that equal access times are guaranteed. Otherwise a disparity in access times necessitates additional logic to equalize time differences.

Loading routes on an array of IPStash devices is equally easy. Upon arrival of a new update the prefix is presented to all IPStash devices in parallel. The devices respond with a hit or miss signal on the arbitration bus depending on whether they can accommodate the new prefix without a conflict. The device with the highest statically-assigned priority gets to store the prefix. If all the devices experience a conflict, the IPStash array is considered to be “full.”

5.3. Route Pruning

Liu has shown that given a routing table, there will be some structure we can exploit to reduce its size [23]. He used two techniques to achieve this: i) mask extension, and ii) pruning which is relevant in our case. An example pruning is shown in Figure 8 where the routing table is organized as a tree structure. The parent of prefix P2 is the longest prefix that matches the first few bits of P2 so P2 is redundant because either P1 or P2 yield the same port number and when P2 is matched, P1 is also matched. Thus, P2 can be removed without affecting the routing functionality. Liu reports that routing tables can be pruned up to 26.6%.

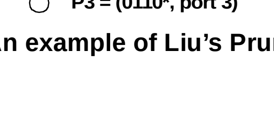

Figure 8. An example of Liu’s Pruning

Techniques similar to pruning can be applied in IPStash since route expansion itself creates many redundant prefixes. In our case, Liu's algorithm can be applied *off-line* by an external agent since it requires access to the entire routing table. Liu's pruning also affects deletions which should be handled as in [23].

Liu's pruning is not required for the successful operation of IPStash but it is merely a size optimization. Since it is a detriment to the simplicity of IPStash it should be avoided for all but the most cost-sensitive applications where the last bit of effective capacity matters. In contrast to Liu's off-line pruning, we apply internal pruning on-line as the routing table is inserted in the IPStash. Internal pruning occurs only among routes conflicting in the IPStash and not on the entire table.

Internal on-line Pruning Algorithm—This type of pruning is intended to reduce the number of conflicts by allowing long prefixes to replace short prefixes when both are expanded to identical lengths.

For example, assume that the prefixes appearing in Figure 8 belong to the same class whose boundaries are bits 2 to 4. Some of the resulting expanded prefixes (of nodes P1 and P2) have exactly the same tag and index even though they come from prefixes of different lengths. In this case, we keep only the expanded prefix that preserves the correctness of the routing table: the one generated by the longest prefix.

We note here two key differences to Liu's pruning: i) we do not discard longer routes in favor of smaller ones, and ii) the port number is irrelevant for deciding which expanded route to discard.

Deletions in an internally-pruned table—When internal pruning is performed, we cannot simply delete entries from the routing table because we might leave holes in the coverage of other shorter prefixes. Deletion in this case becomes a two-step process to patch the holes. The first step is to find out which is the longest prefix in the *in the same class* that matches the deleted route. This requires a search for progressively shorter prefixes in the same class that match the deleted prefix. The second is to modify —instead of delete— the expanded prefixes of the deleted route to become expanded prefixes of its longest match.

In the face of this tedious operation, internal pruning is not appropriate for high-performance applications. The trade-off here is between capacity and update speed: for a small decrease in effective capacity the update speed can be maintained at very high levels. In contrast, in cost-sensitive applications (presumably low-end), one can optimize for capacity and pay a small penalty in update speed.

Pruning results—The effects of the pruning algorithms (both Liu's and internal) are shown in Table 5 where we can see that there is a 10-15% reduction of the expanded routes. This size reduction of the *expanded* routing table corresponds to a reduction in the required maximum (skewed) associativity analogous to that of Section 4.

		PRUNED ROUTES			MAX
	EXPANDED ROUTES	INTERNAL PRUNED (%)	LIU'S (%)	COMBINED (%)	ASSOC. (% DIFF)
RT1	102550	3153 (3)	7545 (7)	10037 (10)	27 (-10)
RT2	194541	7957 (4)	21513 (11)	28539 (15)	47 (-13)
RT3	202094	8352 (4)	18992 (9)	26168 (13)	49 (-13)

Table 5: Pruned Tables

5.4. Reducing uncertainty of effective capacity

Even with pruning, we cannot guarantee that IPStash will accommodate a routing table close to its capacity (when the mean associativity required by the table is much closer than 3σ from the hardware associativity). Many interesting solutions exist to increase the likelihood that a table will fit. We list here three categories for solutions without expanding further in this paper:

- Increasing the apparent associativity: Techniques such as Hash-rehash [1], Column-associativity [2] and others have been proposed to make direct-mapped caches appear set-associative. Similar techniques can also be applied in IPStash to resolve conflicts. IPStash already may require multiple accesses to retrieve an item but such techniques would add further to the latency and power consumption for the few cases that need it.
- Victim caches [19]. This is a classical solution to accommodate conflict-evicted items in direct-mapped caches. In our case an analogous "victim cache" is included and sized to capture the small part of the routing table that is unlikely to fit (as implied by Table 4). In our case, the victim cache is a small TCAM for longest-prefix match. A small TCAM does not consume significant power, but it is searched in parallel with the main IPStash on every access.
- Finally, one can add a second IPStash device in parallel to the first, increasing total associativity. We do not recommend this for a few conflicts since the incremental step in capacity is quite large; rather we use multiple IPStash devices to store significantly larger routing tables as described in Section 5.2.

6. Power Consumption Results

6.1. Cacti

We used the Cacti tool [41] to estimate performance and power consumption of IPStash. Cacti takes as input the characteristics of the cache and iterates over multiple configurations until it finds a configuration optimized for speed, power, and area. Cacti divides the memory array of the cache into several subarrays trying to make them as close to a square as possible to balance the latency of wordlines and bitlines. As such, Cacti puts emphasis on latency considering power as a secondary objective. Thus, results in this section might not be optimal in terms of power consumption and better subarray configurations may exist.

For a level comparison we examined IPStash in the same technology as most of the newest TCAMs. We used technology integration of 0.15μ and we consider 6 bits of output port information as the data payload. With a 12-bit index, the tag is 20 bits (as required by Class 0 tags) plus 5 bits for the original unexpanded length. Finally, power results are normalized for the same throughput —e.g., 100 million searches per second (Msps), a common performance target for many companies— instead of the same frequency. Thus, the operational frequency of the IPStash may not be the same as that of other TCAMs.

Two modifications are needed in order to correctly simulate IPStash. The first is the extra logic (additional decoders and comparators) required for the unexpanded length arbitration which add both latency and power.

Using Cacti's estimates we computed the extra latency added by the length comparators to be less than 0.3ns —without affecting cycle time in pipelined designs— and the power to be less than 0.05W at the highest operating frequency. The second is support for skewed associativity. Skewed index construction (rotations and XORs) introduce negligible latency and power consumption to the design. However, a skewed-associative IPStash requires 8 separate decoders for the wordlines —something Cacti did not do on its own. We computed latency and power overhead of 8 separate decoders when dividing wordlines into 8 segments. We concluded that the skewed-associative IPStash is slightly faster than a standard IPStash while consuming about the same power. The reason is that the 8 decoders required in the skewed-associative case are faster than the monolithic decoder employed in the standard case (which also defines cycle time). At the same time, although each of the small decoders consumes less power than the monolithic decoder, 8 of them together consume slightly more power in total.

Table 6 shows the results for 4 IPStash configurations ranging in capacity from 128K to 1M entries. In IPStash we increase associativity in order to increase size. This is because larger routing tables require higher associativity and for the range of sizes we examined (from 50K to 200K entries) the relation of size and associativity is linear (Section 4). We have extended Cacti to handle more than 32-ways, but as of yet we have not validated these extensions. 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 to approximate higher associativity but we do not account for possible additional routing overheads. Results from our modified Cacti that supports high associativity are better than the ones obtained by the multi-banked approximation.

Cacti shows that IPStash devices in the range of 128K to 1M entries can run up to 750MHz (about 600MHz for the 1M-entry) and easily exceed 100Msps which is the current top-of-the-line performance for TCAMs. All the devices have comparable access times of 3–5ns, and pipelined cycle times of 1.4–1.7ns. We assume that IPStash has a 3-cycle pipeline (1.5ns cycle for the 128K, 256K, and 512K entries, 2ns cycle for the 1M entries). The maximum search latency would be that of three pipelined accesses (a Class 3 match) which

IPStash	32-way x 1	32-way x 2	32-way x 4	32-way x 8
Entries	128K	256K	512K	1M
Access Time (ns)	3.34	3.27	3.56	5.1
Cycle Time (ns)	1.38	1.34	1.33	1.69
Max Search LAT. (ns) (3 pip. accesses = 5c)	5.52	5.36	5.32	6.76
Max Search throughput pip.(Msps)	241	249	251	197
Max Freq. (MHz)	725	746	752	592
Power at 150 MHz, 50Msps (Watts)	0.32	0.64	1.36	3.26
Power at 200 MHz, 66Msps (Watts)	0.43	0.85	1.81	4.34
Power at 250 MHz, 83 Msps (Watts)	0.54	1.07	2.26	5.42
Power at 300MHz, 100 Msps (Watts)	0.65	1.28	2.71	6.51
Power at 500MHz, 166Msps (Watts)	1.08	2.13	4.52	10.86
Power at 600MHz, 200 Msps (Watts)	1.29	2.56	5.42	—
Power at 700MHz, 233 Msps (Watts)	1.5	2.98	6.33	—

Table 6: Cacti's power and timing results

corresponds to 5 cycles: three cycles for the first access plus one for each additional access. This gives us a range of 7.5–10ns for the maximum search latency and a range of ~200Msps to 250Msps for the search throughput.

Power consumption for 100Msps–level performance starts at 0.65W for the 128K-entry device and increases almost linearly with size (1.27W, 2.71W and 6.51W for the 256K, 512K and 1M-entry devices respectively). The results are analogous for the 200Msps level performance.

6.2. Traffic

As we have mentioned, the concept for longest prefix match in IPStash is to iteratively search the set-associative array for progressively shorter prefixes until a match is found. The number of classes and their boundaries determine not only the size of IPStash but also its performance. Size requirements are dictated by the prefix distribution of the routing tables. Performance, in turn, is determined by the percentage of hits per prefix length. As more incoming addresses hit on Class 1 (the first class searched), fewer memory accesses are required, the average search latency is reduced, and less power is consumed.

To evaluate the performance of IPStash on real searches, we obtained three traffic traces collected by the NLANR MOAT team [33]. These traces record the traffic from NASA Ames to the MAE-West router and were collected over OC12c links. Because these traces do not contain a specific routing table snapshot, we used MAE-West routing tables (MW1, MW2 and MW3) captured by the RIPE network coordination center [35]. We selected the routing table snapshot dates to

IEEE COMPUTER SOCIETY

be as close as possible to the capture dates of the traffic traces so we can have realistic results (Table 7).

	Routing Table Size	Traffic Trace Size	Date
Simul. 1 (MW1)	117685	1994855	June 15, 2002
Simul. 2 (MW2)	125256	645147	Oct. 15, 2002
Simul. 3 (MW3)	224736	594651	March 1, 2003

Table 7: Characteristics of the three packet traces and the associated routing tables

Feeding these traffic traces through an IPStash simulator loaded with the appropriate routing table gives us the percentage of packets that match each prefix length. Figure 9 shows these percentages averaged over the three simulations. Given a specific assignment of prefix lengths to classes we can compute a single factor, the average number of accesses per search which characterizes search latency— a Class 1(0) match requires a single access, a Class 2 match one more, and a Class 3 match another (a total of 3 accesses). For the class definition discussed in Section 2.2 and the distribution of matches in Figure 9 this factor is equal to 2.55 accesses/search.

This result was unexpected: the bulk of the prefixes are 24-bit long, yet the small number of 8-bit prefixes (about 1% of the total) gets a fifth of the hits skewing heavily the average number of accesses/packet towards high values. We believe that the traffic we examined gives us conservative results although we have no concrete data to back this up. Our theory is that most of the NASA Ames traffic is destined for remote sites and hits on 8-bit prefixes just to get directed to the next router. Only a small percentage of the traffic hits on 24-bit prefixes that represent entities in MAE-West's immediate network vicinity. We believe that if we had access to the aggregate MAE-West traffic we would see many more packets using 24-bit prefixes as the router would distribute incoming traffic from the rest of the world back to organizations in its immediate network vicinity.

Despite what we think is unrepresentative traffic, we can change class boundaries so we can move to a lower memory access factor. For example, we can change Class 1 to bits 19:24 which yields an average of 1.9 accesses/packet for the same traffic. However, this affects the size of the expanded routing tables (often in a dramatic way —see Figure 3) and consequently power consumption which is also proportional to memory size. In general, the memory access factor varies much less than the memory size factor. Class definition must balance increased size with a lower accesses/packet average to obtain optimal power consumption for a desired search throughput.

Table 8 shows how the coefficient number of 2.55 accesses/packet affects the power and timing results of IPStash devices. Since the routing tables we use in these simulations are on the order of 128K and 256K entries, our results are primarily representative for the 256K- and 512K-entry IPStash devices; we simply extrapolate for the 128K-entry and 1M-entry devices.

Comparing these results with some of the best published industry data (Figure 10) we see that even under unfavorable conditions IPStash easily exceeds the performance of the best products providing more than double the search throughput. Furthermore, IPStash power consumption at its highest is at the level of the announced minimum power consumption of the best TCAM with full power management. Although details for this level of power consumption have not been disclosed by companies, such approaches typically require optimal partitioning of routing tables and external hardware to selectively power-up TCAM blocks. In contrast, IPStash achieves these levels of power consumption by default, without the need for any additional external hardware or effort.

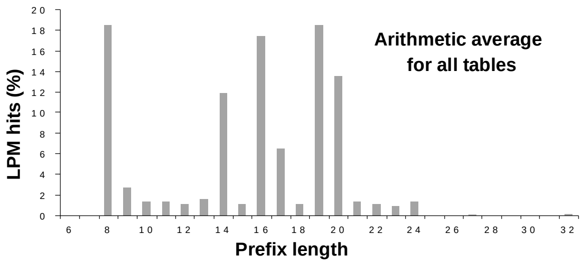

Figure 9. Hits per prefix length

IPStash	32-way x 1	32-way x 2	32-way x 4	32-way x 8
Entries	128K	256K	512K	1M
Access Time (ns)	3.04	2.97	3.26	4.8
Cycle Time (ns)	1.38	1.34	1.33	1.69
Aver. Search latency	3.52	3.42	3.39	4.31
Aver. Search throughput (pipelined) (Msps)	284	292	295	232
Power at 128MHz, 50Msps (Watts)	0.28	0.55	1.16	2.78
Power at 170MHz, 66Msps (Watts)	0.37	0.72	1.54	3.69
Power at 213MHz, 83 Msps (Watts)	0.46	0.91	1.93	4.62
Power at 255MHz, 100 Msps (Watts)	0.55	1.09	2.31	5.54
Power at 425MHz, 166Msps (Watts)	0.91	1.81	3.84	9.23
Power at 510MHz, 200 Msps (Watts)	1.1	2.17	4,61	11.0

Table 8: Cacti's power and timing results for the NASA Ames traffic

7. Related Work

The use of TCAMs for routing table lookup was first proposed by McAuley and Francis [27]. 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 [37]. Kobayashi et al. suggested asso-

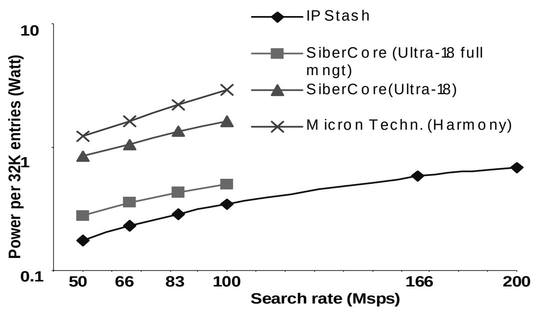

Figure 10. Overall comparison with the top-of-the- line TCAMs

ciating each TCAM entry with a priority [20], and additional hardware was used to output the entry with the highest priority in case of multiple matches. This eliminated the sorting requirement, but added extra latency to lookup operations.

The problem of power consumption in TCAMs was studied by Liu [23]. He used a combination of pruning techniques and logic minimization algorithms to reduce the size of TCAM-based routing tables. Another method proposed a pre-processing step to reduce the size of the required TCAM [25]. Zane et al. [43] take advantage of the effort of several TCAM vendors to reduce power consumption by providing mechanisms to search only a part of the TCAM device.

The idea of prefix expansion was initially introduced by Srinivasan and Varghese [39]. They used this technique to reduce the depth of a trie-based routing table. Since then, many researchers used this technique to reduce lookup time in their software trie-based algorithms [31,22].

Many researchers used caches to speed up the translation of the destination addresses to output port numbers. Results for Internet traffic studies [32,9,7] showed that there is a significant locality in the packet streams. Estrin and Mitzel [11] derive the storage requirements for maintaining state and lookup information on the routers, and showed that locality exists by performing trace-driven simulations.

Most software-based routing table lookup algorithms try to optimize the usage of the cache in general purpose processors, such as algorithms proposed in [10] and [22]. The approach is to design data structures whose entries fit in a cache line, or fit several entries into the same cache line to allow several lookups per memory access. Chiueh et al. [9,10] proposed two schemes to improve performance of IP address caching.

Talbot et al. [40] studied several traces captured from different access routers. A profiling of the address bit frequency determined which bits should be used for cache indexing. Caching based on carefully selected indexing bits showed very good locality in the IP traces captured. This result highly depends on the choice of indexing bits, which in turn, depends on the trace. A fixed hardware implementation might prove to be ineffective if traffic conditions change.

Cheung et al. [8] formulated the problem of assigning part of routing table to a different cache hierarchy as a dynamic programming problem, and introduced a placement algorithm to minimize the overall lookup speed. Besson et al. [5] also evaluated hierarchical caches in routers and their effect on IP-lookup. Jain [18] studied cache replacement algorithms for different types of traffic (interactive vs. non-interactive). Pink et al. [6] proposed a technique to compress an expanded trie representation of a routing table, so that the result is small enough to fit in the L2 cache of general purpose processor.

Liu [24] recently proposed the prefix cache. The author takes advantage of the natural hierarchy of routing prefixes. Instead of caching an IP address or an arbitrary portion of it, a routing prefix designated by network operators is cached. The author uses a number of algorithms, which differ mostly on how they transform the routing table so that correct results are always guaranteed. The prefix cache is a fully associative cache, which stores the prefix and the mask bits at the tag side and the destination port information at the data side. Simulation results showed that this design works better than caching full IP addresses (fixed 32-bit IP addresses), even after factoring in the extra complexity involved.

Our approach is different from all previous work. Instead of using a cache in combination with a general-purpose processor or an ASIC routing engine, we are using a stand-alone set-associative architecture. We categorize routing prefixes into classes and we use a controlled prefix expansion technique for each category. The expanded prefixes are placed into the IPStash using a portion of the prefix as index. The rest of the prefix is stored as tag along with its original unexpanded length (so that correct lookup results are guaranteed). The data side of IPStash contains the output port information. IPStash offers unparalleled simplicity compared to all previous proposals while being fast and power-efficient at the same time.

8. Conclusions

In this paper, we propose a set-associative architecture called IPStash to replace 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, only their blocks are too few, their associativity is too high, and their comparators are embedded in the storage array instead of being separate.

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 for shorter prefixes. To bound the number of iterations we expand prefixes to a set of predefined lengths. These fixed-lengths stem from the distribution of prefixes in actual routing tables. We used skewed associativity to maximize the efficiency of the limited number of associative ways available. In addition, pruning techniques can be used on top of this, but they tend to complicate other IPStash functionality such as deletions.

Using Cacti and traffic simulations, we study IPStash for current routing table sizes and find that it can be twice as fast as the top-of-the-line TCAMs while offering about 35% power savings (for the same throughput) over the announced minimum power consumption of commercial products. In addition, IPStash can exceed 200 Msps while the state-of-the-art performance for TCAMs (in the *same* technology) currently only reaches about 100 Msps.

Given the current developments in TCAMs we believe that IPStash is a natural next step for large-scale IP-lookup using associative memories. Furthermore, we believe that IPStash can be expanded to many other applications such as IPv6, NAT, the handling of millions of "flows" (point-to-point Internet connections) by using similar techniques as in our proposal.

9. Acknowledgments

This work is supported by Agere Systems Inc. We are grateful for the helpful discussions with Nevin Heinzte and Dan Wang of Agere Systems. We also wish to thank the reviewers for their many useful suggestions.

10. References

[1] A. Agarwal, M. Horowitz, J. Hennesy, "Cache Performance of Operating Systems and Multiprogramming Workloads." *ACM Transactions on Computer Systems*, Nov. 1988.
[2] A. Agarwal and S. D. Pudar, "Column-associative Caches: a Technique for Reducing the Miss Rate of Direct-mapped Caches," *ISCA-20*, May 1993.
[3] F. Baboescu, S. Singh, G. Varghese, "Packet Classification for Core Routers: Is there an alternative to CAMs?" *IEEE INFOCOM*, April 2003.
[4] A. Basu, G. Narlikar, "Fast Incremental Updates for Pipelined Forwarding Engines," *IEEE INFOCOM*, April 2003.
[5] E. Besson, and P. Brown, "Performance Evaluation of Hierarchical Caching in High-Speed Routers." Proc. *Globecom*, pp. 2640-45, 1998.
[6] A. Brodnik, S. Carlsson, M. Degermark, S. Pink, "Small Forwarding Tables for Fast Routing Lookups." *ACM SIGCOMM*, September 1997.
[7] X. Chen, "Effect of Caching on Routing-Table Lookup in Multimedia Environments." *IEEE INFOCOM*, April 1991.
[8] G. Cheung and S. McCanne, "Optimal Routing Table Design for IP Address Lookups Under Memory Constraints." *IEEE INFOCOM*, pp. 1437-44, 1999.
[9] T. Chiueh and P. Pradhan, "High Performance IP Routing Table Lookup Using CPU Caching." *IEEE INFOCOM*, April 1999.
[10] T. Chiueh and P. Pradhan, "Cache Memory Design for Network Processors." *Proc. High Performance Computer Architecture*, pp. 409-418, 1999.
[11] D. Estrin and D. Mitzel, "An Assessment of State and Lookup Overhead in Routers." *IEEE INFOCOM*, May 1992.
[12] EZ Chip Network Processors. http://ezchip.com
[13] A. Gallo, "Meeting Traffic Demands with Next-Generation Internet Infrastructure." *Lightwave*, 18(5):118-123, May 2001.
[14] G. Huston, "Analyzing the Internet's BGP Routing Table." *The Internet Protocol Journal*, 4, 2001.
[15] IBM PowerNP Network Processors. http://www.chips.ibm.com/products/wired/network_processors.html
[16] IDT. http://www.idt.com
[17] Intel IXP2850 Network Processor. http://www.intel.com/design/network/products/npfamily/ixp2850.htm
[18] R. Jain, "Characteristics of Destination Address Locality in Computer Networks: a Comparison of Caching Schemes." *Computer Networks and ISDN Systems*, 18(4):243-54, May 1990.
[19] Norman P. Jouppi, "Improving Direct-Mapped Cache Performance by the Addition of a Small Fully-Associative Cache and Prefetch Buffers." *ISCA-17*, pp. 364--373.
[20] 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.
[21] C. Labovitz, G.R. Malan, F. Jahanian, "Internet Routing Instability." The *IEEE/ACM Transactions on Networking*, Vol. 6, no. 5, pp. 515-528, 1999.
[22] B. Lampson, V. Srinivasan, G. Varghese, "IP-lookups Using Multiway and Multicolumn Search." Proceedings of *IEEE INFOCOM*, vol. 3, pages 1248-56, April 1998.
[23] H. Liu, "Routing Table Compaction in Ternary CAM." *IEEE Micro*, 22(1):58-64, January-February 2002.
[24] H. Liu, "Routing Prefix Caching in Network Processor Design." *IEEE ICCCN2001*, October 2001.
[25] J. van Lunteren and A.P.J. Engbersen. "Multi-Field Packet Classification Using Ternary CAM." *Electronics Letters*, 38(1):21-23, 2002.
[26] R. Mahajan, D. Wetherall, T. Anderson, "Understanding BGP Misconfiguration." *SIGCOMM '02*, August 2002.
[27] A. J. McAuley and P. Francis, "Fast Routing Table Lookup Using CAMs." In *Proceedings of INFOCOM '93*, pages 1382-1391, San Francisco, CA, March 1993.
[28] Micron Technology. http://www.micron.com
[29] Netlogic microsystems. http:// www.netlogicmicro.com
[30] Network and Communications ICs. http://www.agere.com/enterprise_metro_access/network_processors.html
[31] 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.
[32] C. Partridge, "Locality and Route Caches." *NSF Workshop on Internet Statistics Measurement and Analysis* (http://www.caida.org/ISMA/Positions/partridge.html), 1996.
[33] Passive Measurement and Analysis project, National Laboratory for Applied Network Research. http://pma.nlanr.net/PMA
[34] Y. Rekhter, T. Li, "An Architecture for IP Address Allocation with CIDR." RFC 1518, Sept. 1993.
[35] RIPE Network Coordination Centre. http://www.ripe.net
[36] A. Seznec, "A case for two-way skewed-associative cache," Proceedings of the 20th *International Symposium on Computer Architecture*, May 1993.
[37] D. Shah and P. Gupta, "Fast Updating Algorithms for TCAMs." *IEEE Micro*, 21(1):36-47, January-February 2001.
[38] Sibercore Technology. http://www.sibercore.com
[39] V. Srinivasan and G. Varghese, "Fast Address Lookups Using Controlled Prefix Expansion." *ACM Transactions on Computer Systems*, 17(1):1-40, February 1999.
[40] B. Talbot, T. Sherwood, B. Lin, "IP Caching for Terabit Speed Routers." *Global Communications Conference (Globecom'99)*, pp. 1565-1569, December, 1999.
[41] Steven J. E. Wilton and Norman P. Jouppi, "Cacti: An Enhanced Cache Access and Cycle Time Model." *IEEE Journal of Solid-State Circuits*, May 1996.
[42] Vitesse IQ2200. http://www.vitesse.com
[43] F. Zane, G. Narlikar, A. Basu, "CoolCAMs: Power-Efficient TCAMs for Forwarding Engines." *IEEE INFOCOM*, April 2003.

IEEE COMPUTER SOCIETY