IPStash: a Power-Efficient Memory Architecture for IP Lookup

Stefanos Kaxiras[*] and Georgios Keramidas[+]

[*]Agere Systems

[+]University of Patras, Department of Electrical and Computer Engineering

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, in this paper, 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 pruning techniques and* skewed associativity *to increase the effective capacity of IPStash 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 up to* 600 *MHz, offer up to twice the search throughput (200Msps), and consume u*p *to 40% 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 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) standard [36].

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 [30], EZChip's NP-1 [32], Agere's APP550 [33], IBM's PowerNP (NP4GS3) [31] and Vitesse's IQ2200 [34] 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 longest-prefix match [8][11][12][14][19]. 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 routes ending with wildcards —called *route prefixes* or simply *prefixes*— 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 matches anything. Several companies (IDT [26], Netlogic [27], Micron [28], Sibercore [29]) currently

offer a large array of TCAM products used in IP lookup.

TCAMs automatically report the topmost match (out of many) so longest prefix match is guaranteed only if the routing table is sorted and loaded correctly: long route prefixes must appear before short ones. This leads to difficult problems in updating routing tables in TCAMs and significant effort has been devoted in addressing these problems [3][4].

TCAMs are an attractive technology for IP route lookup operations. But two major drawbacks hamper their wide deployment: 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 [26][27]. TCAM power consumption is critical in router applications because it affects to two important router characteristics: *linecard power* and *port density.* Linecards have fixed power budgets because of cooling and power distribution constraints [23]. 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 relevant blocks have reduced power consumption considerably [8][26][27]. 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 external routing table management requiring not only correct sorting but also correct partitioning. 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.

IPStash—To address TCAM problems we propose a new memory architecture for IP lookup we call *IP-Stash*. 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 (Section 3.4). IP-Stash 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 route table updates: since the routing table needs no special handling, updates are also straightforward to perform. Updates are simply writes to the IPStash.
- Low power: Accessing set-associative memory is far more power-efficient than accessing 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 an SRAM memory requires 4-6 transistors. Even when TCAMs are implemented using DRAM technology they can be less dense than SRAMs or take more area.
- 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.

Contributions of this paper—The contributions of this paper with respect to IPStash and IP lookup 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 this architecture. Since we do not support "don't care" bits, we allow only a limited number of different prefix lengths in the IP-Stash. 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 apply route compaction algorithms which are efficiently implemented in our architecture. Moreover, we show how *skewed associativity* [5] can be applied with great success to increase effective capacity.
3. We use real data to validate our assumptions with simulations. We use a modified version of the Cacti tool to estimate power consumption and we show that IP Stash consumes up to 40% less power than the best commercial available blocked TCAMs.

Structure of this paper—Section 2 presents the IPStash architecture and its implementation of the longest prefix match algorithm. In Section 3 we analyze the techniques we use (route pruning and skewed associativity) to increase the effective capacity of our devices while in Section 4 we discuss some special cases and the expandability of the architecture. In Section 5 we present simulation results for power consumption. Finally Section 6 summarizes related work and Section 7 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.*

In this section we discuss longest prefix match, how it is performed in TCAMs, and how it can be performed in IPStash exploiting the distribution of prefix lengths in routing tables.

2.1 Routing tables and longest prefix match

Since the advent of the Classless Inter-Domain Routing (CIDR) in 1993 [36], IP routes have been identified by a <route prefix, prefix length> pair, where the prefix length is between 0 and 32 bits. For every incoming packet, a search must performed in the router's forwarding table to determine which next hop the packet is destined for. With CIDR, the search may be decomposed into two steps. First, we find the set of routes with prefixes that match the beginning of the incoming IP destination address. Then, among this set of routes, we select the one with the longest prefix. This is the route that we use to identify the next hop.

Various software-based schemes have been proposed for this search problem [11][12][14] but in many situations lack the necessary speed and/or simplicity. With increasing requirements for high throughput, hardware implementations of algorithmically simpler solutions such as fully-associative search have found acceptance in commercial products. A prime example is the TCAM. The ternary capability allows a TCAM to store variable length prefixes by storing "don't cares." Lookups are performed in a TCAM by storing routing table entries in order of decreasing prefix lengths and choosing the first entry among all the entries that match the incoming packet destination address.

Consider the example in Figure 1 and assume for simplicity that the maximum route length is 4. An incoming packet (with destination address 0011) arrives at the TCAM. The TCAM compares the destina-

tion address of the incoming packet against all the prefixes in parallel. Two matches occur: the entries at the memory locations 1 and 2. A priority encoder then selects the first matching entry—the entry with the lowest address (1). If we load the routing table correctly, this should give us the longest prefix match.

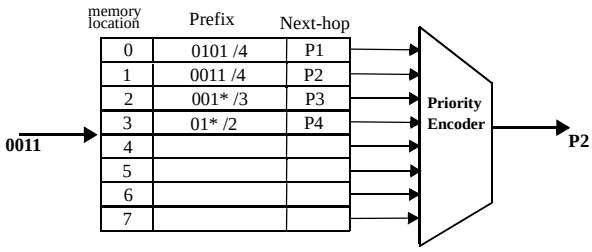

FIGURE 1. Routing table in a TCAM

The need to maintain a sorted routing table in the 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. Much better algorithms have been proposed, however all require an external entity to manage and partition the routing table. IPStash on the other hand does not implement "don't care" bits. This necessitates a different approach for longest prefix match, which we describe in the next section.

2.2 Longest Prefix Match in IPStash

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. To simplify the problem we consider 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 2) 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 routes 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 2:

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 route —the length of the route 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 route to match. Again we access the appropriate set and we search again the 32 possible results but now for a length-20 match and for an 8-bit tag.
3. Step 3: Similarly for the last case we try to match a 16-bit route 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. Moreover, as we will explain in subsequent sections, the length match can actually be a search for the longest length rather than a simple comparison.

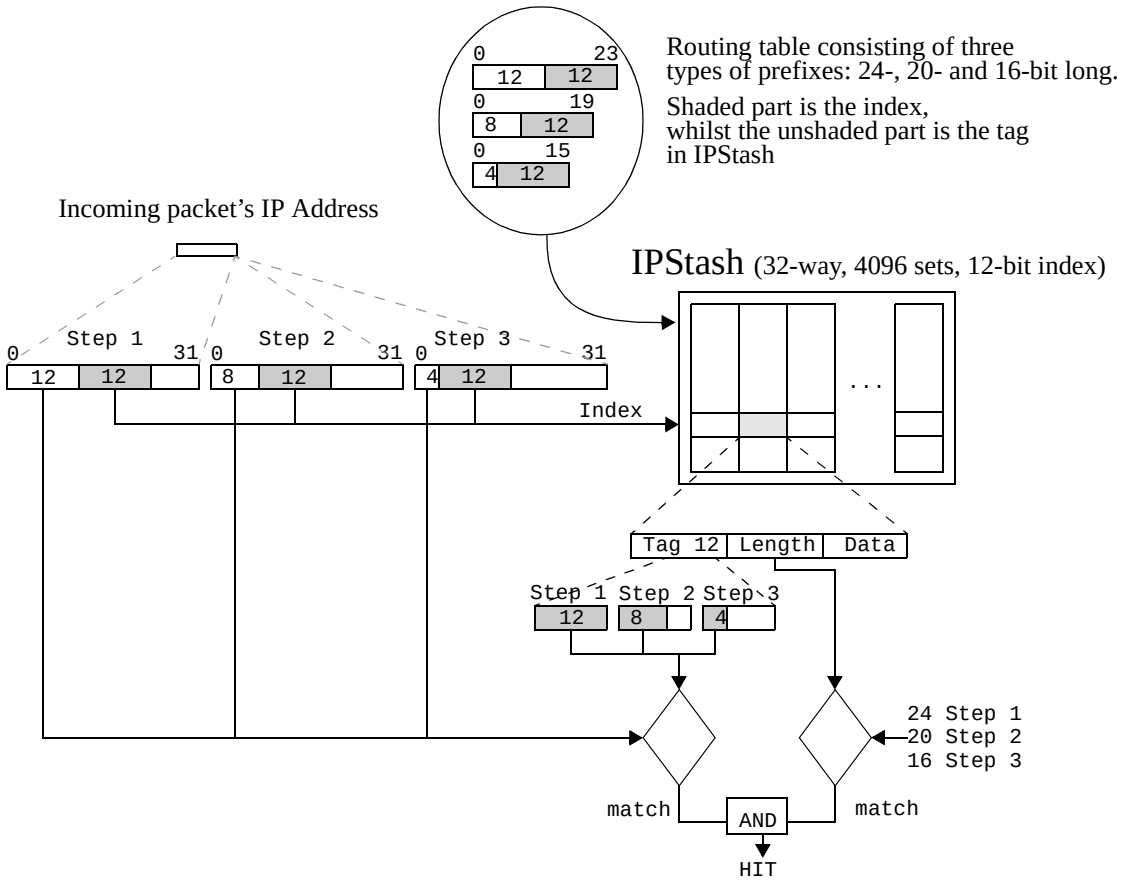

FIGURE 2. IPStash operation for three prefix lengths.

2.3 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 all the prefixes of different lengths to a small set of fixed lengths. The choice of which prefixes to expand and how much depends 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 [35][11][12][14] 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 [24].

Figure 3 shows the distribution of prefix lengths for the three indicative tables we use in this paper (log-scale view on top for clarity and normal view below). We can easily draw conclusions —also noted by other researchers— from the graphs in Figure 3: 24-bit prefixes comprise about 60% of the tables; prefixes more 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.

These observations lead to a natural categorization of the route table into five classes (Figure 3):

- **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-bits 2 fold. In addition, in this class we added a "shadow class," Class 0 —explained below— for the prefixes with more

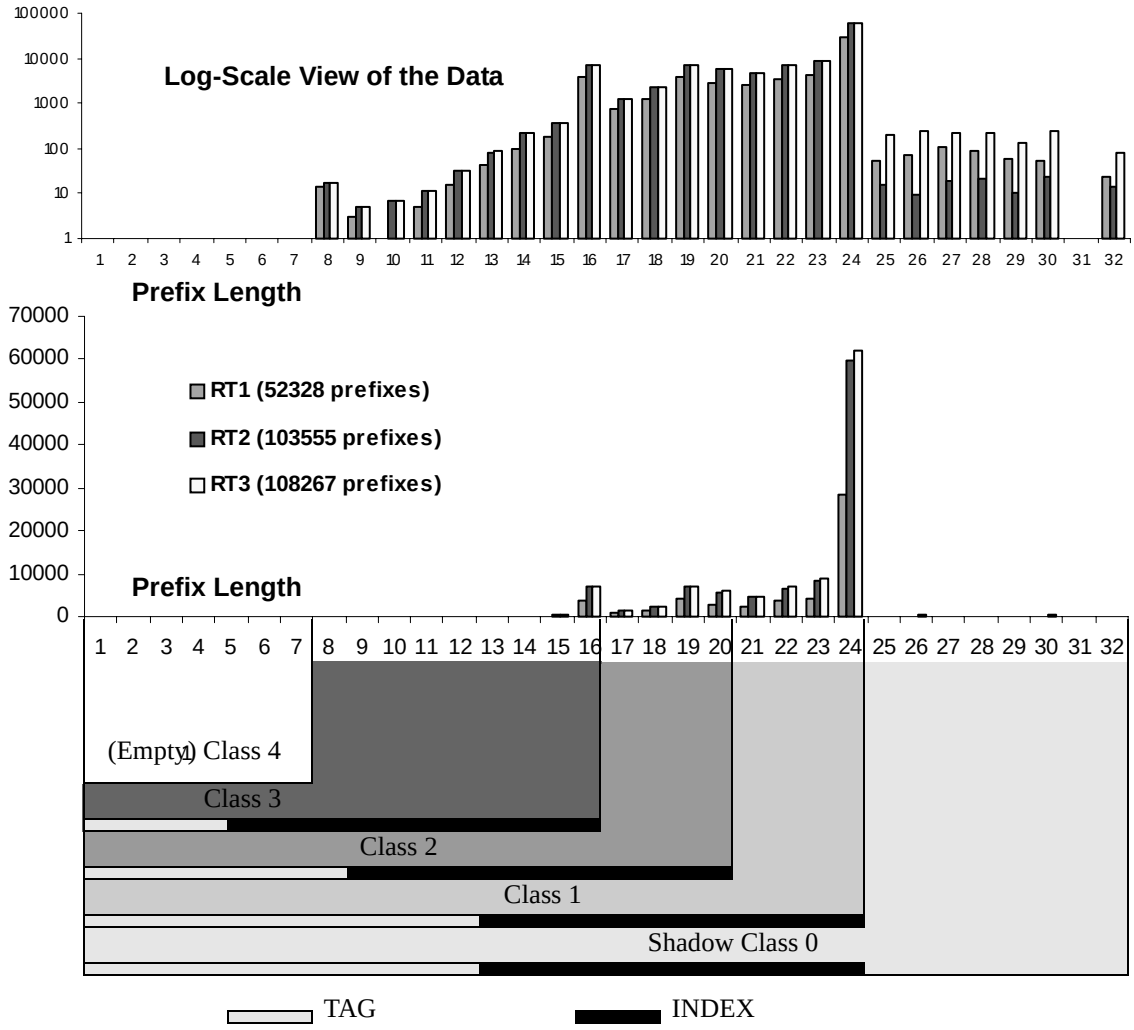

FIGURE 3. Distribution of prefix lengths for 3 routing tables

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, 11-bit 32-fold, and so on) because there are so few of them. Prefixes smaller than 8 bits are typically not found in routing tables.
- **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.[1]
- **Class 0** contains all prefixes longer than 25 bits. This is a special *shadow* class because we fold it on top of Class 1. We use exactly the same index and tag as in Class 1 so Class 0 entries are matched with a Class 1 access. Note that 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

[1] However, if need be, IPStash can handle Class 4 prefixes.

Class 0 entries on top of Class 1 entries has negligible effects. However, this folding necessitates a final length arbitration to determine a hit out of many possible matches. Thus, IPStash determines hits both by tag match and by selecting the longest *unexpanded-length* entry. We take this into account in later sections.

As we mentioned above, the number of classes as well as their bounds are derived from the structure of the routing tables. We envision IPStash as a configurable device where the classes are set during power-up, but for the sake of simplicity we restrict our examples to the aforementioned classes. We examined different class combinations (e.g., 2 classes, spanning bits 8 to 16 and 17 to 24) but found that the natural bounds indicated by the length distributions in Figure 3 offered reasonable results.[2]

Prefix expansion can happen externally or internally in IPStash. External prefix expansion requires the cooperation of the entity using the IPStash (e.g., a network processor) so that only prefixes of the correct lengths —as determined by the configuration— are stored. Internal expansion is straightforward requiring only a small FSM and counters to fill the wildcard bits, but it makes prefix insertion in the 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 the inflation are shown in Table 1 for three actual routing tables RT1, RT2 and RT3:

	Initial routes	Expanded Routes	Increase
RT1	52328	102550	1.96
RT2	103555	194541	1.88
RT3	108267	202094	1.87

Table 1: The effect of the route expansion

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 for example 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.

	Expanded Routes	IPStash Configuration	Unresolved Conflicts
RT1	102550	128K entries: 3840 Kbits	1685 (~3.2%)
RT2	194541	256K entries: 7680 Kbits	566 (~0.55%)
RT3	202094	256K entries: 7680 Kbits	979 (~0.9%)

Table 2: IPStash configurations for routing tables and resulting conflicts

[2] Since classes are searched sequentially for a hit, the average number of accesses per hit is determined by the incoming traffic. This in turn might affect the choice of classes (see the discussion in Section 5.2).

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

3 Increasing effective capacity

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, there are two main approaches to eliminate conflicts: i) by detecting and eliminating redundant prefixes that result from route expansion, and ii) by making better use of the available associativity. For the first case, we perform route pruning similar to that of Liu [6]. For the second we show how Seznec's ideas of skewed associativity [5] can be applied in IPStash with great success.

3.1 Route pruning

Liu has shown that given a route table, there will be some structure we can exploit to reduce its size [6]. Liu used two techniques to achieve this: i) mask extension, and ii) pruning which is relevant in our case. An example of Liu's pruning is shown in Figure 4 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. In this simple example, 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 routing functionality. Liu reports that routing tables can be pruned up to 26.6%.

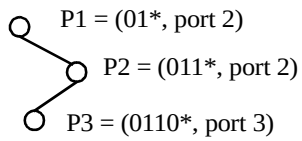

FIGURE 4. An example of Liu's pruning

Techniques similar to pruning are especially well suited in our case since route expansion creates many redundant routing prefixes, thus many extraneous conflicts in IPStash. 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 is not required for the successful operation of IPStash but it is merely a size optimization. In fact, because 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* —automatically and transparently to the outside world— 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 4 belong to the same class whose bounds are bits 2 to 4. This leads to the prefix expansions shown in Figure 5. Some expanded prefixes (shaded) have exactly the same tag and index even though they come from prefixes of different lengths. In this case, we

[3] One could treat IPStash as a cache, however this would introduce very high variance in access latency which is undesirable in a router environment.

keep only the expanded prefix that preserves the correctness of the routing table: the one generated by the longest prefix.

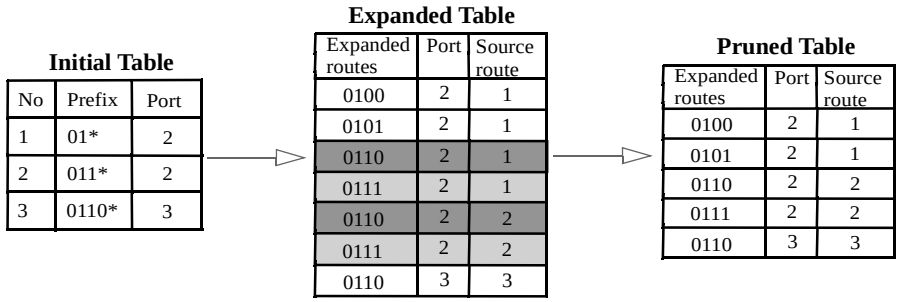

FIGURE 5. Installation of the prefixes

Technically, expanded routes with the same index and tag are not in conflict unless the set containing them is out of space. IPStash *can* handle identical tags within the same set, since the *unexpanded prefix length* is preserved in the tag and length arbitration can resolve multiple hits. In such a case, the IPStash entry generated by the shorter prefix would be inaccessible. By discarding these inaccessible prefixes at insertion time we free up valuable space thus reducing conflicts of entries with different tags.

Finally, 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.

***Pruning results*—**The effects of the pruning algorithms (both Liu's and internal) are shown in Table 3 where we can see that there is a 10-13% reduction of the expanded routes and there is also a significant reduction of the number of unresolved conflicts (especially for the two larger routing tables).

	EXPANDED ROUTES	PRUNED ROUTES			SAVINGS			CONFLICTS			
		INTERNAL PRUNED:	LIU'S	COM-BINED	INTERNAL PRUNED:	LIU'S	COM-BINED:	INTERNAL PRUNED:	LIU'S	COM-BINED	ORIGI-NAL
RT1	102550	3153	7545	10037	3.07%	7.36%	9.79%	1125	1073	525	1685
RT2	194541	7957	21513	28539	4.09%	11.06%	14.7%	281	303	41	566
RT3	202094	8352	18992	26168	4.13%	9.4%	12.94%	521	636	139	979

Table 3: Pruned Tables

3.2 Skewed associativity: fitting more in the same space

Another approach is to try to increase the utilization of the set-associative memory by reducing the number of conflicts to a minimum. For this purpose we turn our attention to Seznec's idea of a skewed-associative cache [5]. 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 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 6 shows how RT3 is loaded into an "unlimited-associativity" IPStash —there is no 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 6) is to smooth-out the jagged edges of the original graph —in some sense to increase the entropy of the system, to increase disorder. Contrary to common sense a flat and homogeneous graph is very disorderly.[4]

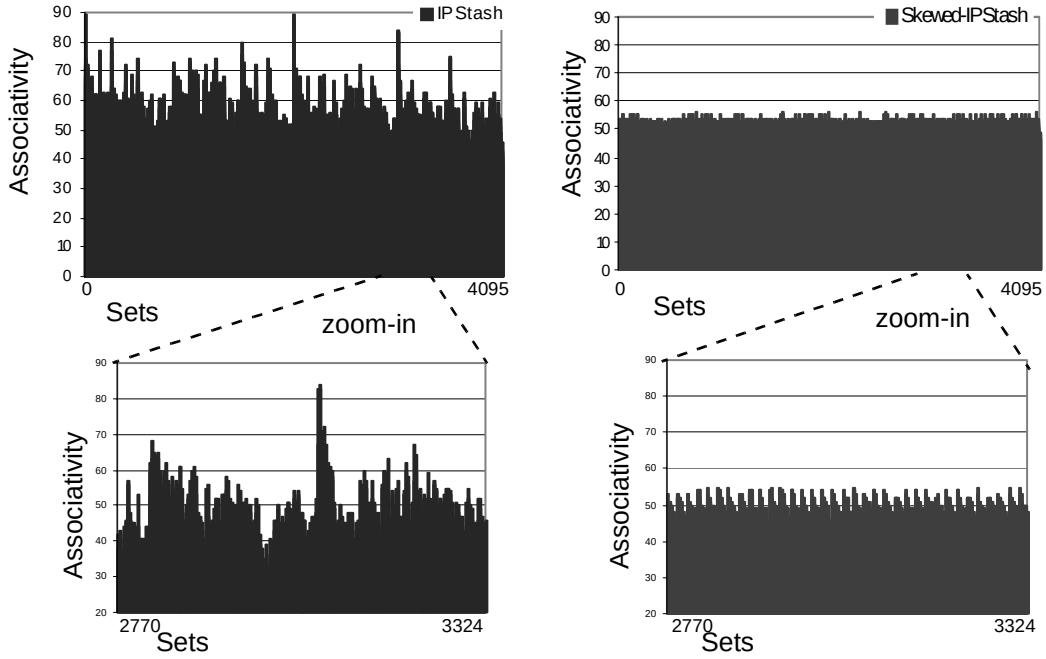

FIGURE 6. Original IPStash and skewed IPStash comparison (for RT3). Skewing is an increase of the entropy of the system.

We produced the skewed-associative graph in Figure 6 using a simple skewing technique. The key issue in IPStash is that we do not have so much freedom to create 32 (or 64) skewing functions because we have only few significant bits to exploit. Our compromising solution is to create only 8 (4 for Class-3 accesses) different skewed indices instead of 32 or 64 as the hardware associativity would indicate. 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 7:

- For Class 1,0, and 2 addresses: 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 addresses: 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

[4] This is analogous to the notion of entropy in Physics: mountains although jagged and quite random in appearance represent order; deserts although flat and homogeneous represent disorder. As entropy increases, mountains turn into deserts.

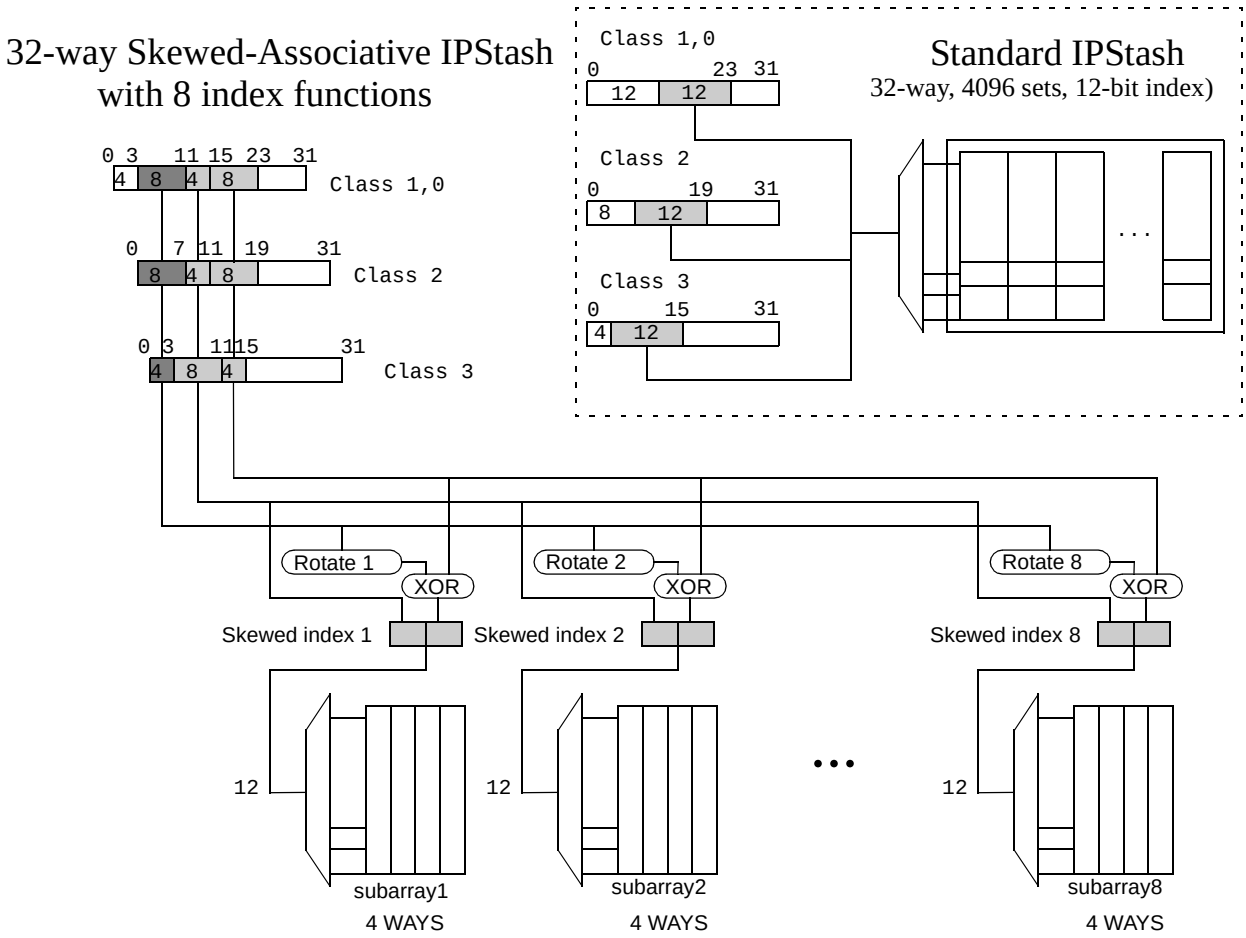

FIGURE 7. Index extraction for skewed associativity

in two different banks.

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 7 is sufficient to illustrate our case. Skewing incurs a small hardware cost since the mapping functions are chosen for their simplicity.

3.3 Conflict Results for Pruning *and* Skewed Associativity

Putting it all together we see that skewed associativity successfully reduces the conflicts to zero with or without pruning for the three routing tables we use. Pruning frees up some more space which can be important if the size of the expanded routing tables critically approaches the size of IPStash. Results presented in this section also give us a rudimentary tool to estimate the capacity of IPStash devices with respect to routing tables.

Table 4 and Table 5 quantify the effects of applying skewed associativity to the IPStash devices. Table 4 assumes that no pruning (either Liu's or internal) was performed on the expanded prefixes while Table 5 includes full pruning. Without skewed-associativity Table 4 shows that a 32-way IPStash would have 1658 conflicts for RT1 —525 if it was pruned— (similarly for RT2 and RT3). This is because as RT1 enters the IPStash each set is filled differently. Some sets only receive 9 entries which is the *Min associativity* required by RT1 while some sets receive 49 distinct entries which is the *Max associativity.* The aver-

age number of entries over all sets is the *Average associativity* required by RT1. While —not surprisingly— the average associativity of the skewed-associative and the standard IPStash is the same for all cases (for RT1, RT2, and RT3, whether pruned or not) 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 6. In our examples the *Max associativity* for the skewed-associative cases does not exceed the hardware associativity of the IPStash in any case.

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 6 shows the results[5] for a CI up to a 0.9999. This method is a convenient tool to check IPStash requirements of individual routing tables or of families of routing tables with similar hashing distribution.

	RT1 on 32-way		RT2 on 64-way		RT3 on 64-way	
Initial Routes	52328		103555		108267	
Expanded Routes	102550		194541		202094	
	Standard	Skewed	Standard	Skewed	Standard	Skewed
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 (mean)	25	25	47.5	47.5	49.34	49.34
Standard Deviation (σ)	5.76	1.65	7.82	2.64	8.09	2.69

Table 4: Skewed IPStash Detailed Comparison without pruning

	RT1 on 32-way		RT2 on 64-way		RT3 on 64-way	
Initial Routes	52328		103555		108267	
Expanded Routes	102550		194541		202094	
After Pruning	92512		169582		175858	
Savings	9.79%		12.83%		12.98%	
	Standard	Skewed	Standard	Skewed	Standard	Skewed
Total Conflicts	525	0	68	0	132	0
Min associativity	9	18	20	35	19	37
Max associativity	44	27	79	47	80	49
Average associativity (mean)	22.59	22.59	41.4	41.4	42.93	42.93
Standard Deviation (σ)	5.24	1.54	6.9	2.41	7.16	2.44

Table 5: Pruned IPStash - Skewed IPStash Detailed Comparison

Std. Deviation range (around mean)	Confidence Interval (Probability of falling inside the range)	RT1 required Associativity		RT2 required Associativity		RT3 required Associativity	
		Standard:	Skewed:	Standard:	Skewed:	Standard:	Skewed:
σ (±0.5σ)	0.6836	28	26	52	49	54	51
2σ (±σ)	0.9543	31	27	56	51	58	53
3σ (±1.5σ)	0.9973	34	28	60	52	62	54
4σ (±2σ)	0.9999	37	29	64	53	66	55

Table 6: Probabilistic associativity requirements

[5] The well known 6σ range would give us an even better CI but we consider a range of up to 4σ (±2σ around mean) to be satisfactory for our application.

Many interesting solutions exist for the cases when the routing table almost fits in the IPStash but not quite, i.e., the cases when the mean associativity required by the routing table is closer than 3σ —for all practical purposes— from the hardware associativity. We list here three broad categories without expanding further in this paper:

- Increasing the apparent associativity: Techniques such as Hash-rehash [39], Column-associativity [40] 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 [38]. 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 6). 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 4.1.

3.4 Associativity as a function of routing table size

Given a skewed-associative IPStash using a 12 bit index (4096 sets) Figure 8 shows the relationship of the required associativity to the *original unexpanded* size for our three example routing tables (RT1, RT2 and RT3) and the MAE-West routing tables (MW1, MW2 and MW3) used in the traffic simulations of Section 5.2. 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 (for the range of sizes studied) which implies that our design scales well.

It is this relationship that justifies our set-associative approach to IP-lookup.

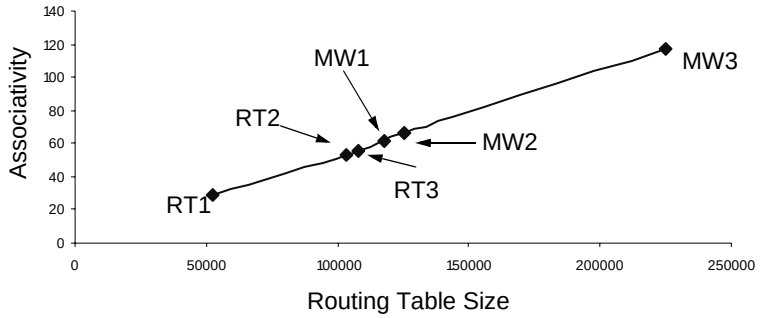

FIGURE 8. Associativity as a function of the routing table size for skewed-associative, 4096-set, IPStash.

4 Other features of the architecture

4.1 Expanding the IPstash

As a result of CIDR, the trend for routing table sizes is a rapid increase over the last few years [23][24]. It is hard to predict routing tables 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.

IP-Stash 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 the existing design. We consider this as one of the main advantages of our proposal. Adding in parallel more IPStash devices increases associativity. Figure 9 shows the proposed scheme.

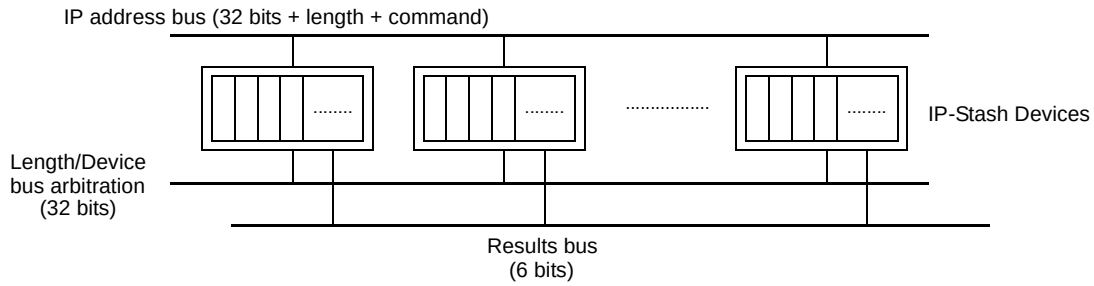

FIGURE 9. Multiple IP-Stash devices

The above scheme resembles a cache-coherent bus. All IPStash devices are on the same (logical) buses. They accept requests (incoming IP addresses, or a prefix with its length, 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). 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). However, 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."

4.2 Updates: route insertions and deletions

The requirement for a fast update rate is essential for a router design. This is true because the routing tables in routers are not static [1]. Many changes in the routing tables occur due to changes in network

topology. In addition, reboots of neighboring routers or BGP misconfigurations [25] 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 network conditions the next generation of forwarding engines requires bounded processing overhead for updates in the face of several thousand —possibly between 5 and 10 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. 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. This extra space, however, leads to fragmentation and reduced capacity.

IPStash does not suffer from such difficult problems because it does not require any significant transformation of the routing tables beyond prefix expansion. Prefix expansion by itself does not affect updates. However, internal pruning of the expanded prefixes complicates deletions as explained below. In the case where Liu's pruning has been performed externally, deletions face the same problems as in [6] and should be handled accordingly (off-line) [6]. This complication reenforces our recommendation for generally avoiding Liu's pruning for IPStash.

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 if no internal pruning is performed.[6] In this case, 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.

The complex case: deletions with internal pruning—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. Consider the example in Figure 10. Prefix *1* (4 expanded entries) and *2* (2 expanded entries) are inserted into the IPStash. Since *1* is itself a prefix of *2*, internal pruning allows *2*'s expanded prefixes to overwrite *1*'s prefixes. The problem now is that it is not safe to delete *2*'s expanded prefixes because this leaves a hole of 2 entries in the set of *1*'s expanded prefixes.

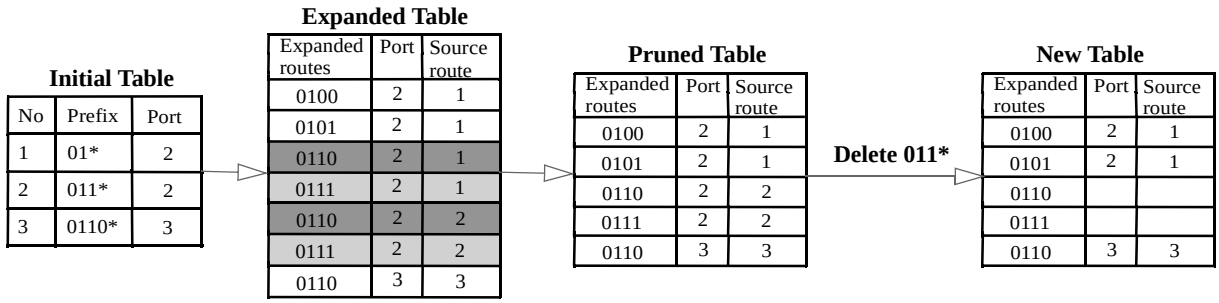

FIGURE 10. Deletion of prefixes with internal pruning

[6] We remind here that IPStash does have the ability to store identical expanded prefixes that were generated from different-length unexpanded prefixes. This works because IPStash arbitrates multiple matches on the unexpanded length. Thus, internal pruning can be switched off at will.

Deletion becomes a two-step process when internal pruning is performed. 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 that an external agent to search for progressively for 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.

5 Power Consumption Results

5.1 Cacti

We used a modified version of the Cacti tool [37] 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 cache 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 0.15u and we consider 11 bits (5 bits for the original unexpanded length and 6 bits for the output port information) as the data payload. The tag is 20 bits using a 12-bit index. 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 more changes are needed in Cacti to simulate IPStash. The first is the extra comparators required for length arbitration. These comparators add both latency and power to each access. 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 0.02W 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 original monolithic decoder, 8 of them together consume slightly more power in total.

Table 7 shows the results for 4 IPStash devices ranging in capacity from 128K-entries 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 over 200K entries) the relation of size and associativity is linear. We have extended Cacti to handle more than 32-ways, but as of yet we have not validated 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 to approximate higher associativity but we do not account for possible additional routing overheads.[7]

IPStash	32-Way	64-Way	128-Way	256-Way
Entries	128K	256K	512K	1M
Bank configuration	32-way x 1	32-way x 2	32-way x 4	32-way x 8
Access Time (ns)	3.04	2.97	3.26	4.8
Cycle Time (ns)	1.38	1.34	1.33	1.69
Max Search latency (ns) (3 pipelined accesses == 5c)	5.52	5.36	5.32	6.76
Max Search throughput pipelined (Msps)	241	249	251	197
Max Frequency (MHz)	725	746	752	592
Total Energy (all banks) (nJ)	2.15	4.26	9.04	21.71
Power at 170MHz, 66Msps (Watts)	0.37	0.72	1.54	3.69
Power at 300MHz, 100Msps (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	—

Table 7: Cacti's power and timing results for the 32-way associativity

With our modifications, Cacti shows that IPStash devices in the range of 128K-entries to 1M-entries can run at more than 700MHz (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 1.4–1.7ns. We assume that IPStash has a 3-cycle pipeline (1.5ns cycle for the 128K-entry, 256K-entry, and 512K-entry, 2ns cycle for the 1M-entry IPStash). The maximum search latency would be that of three pipelined accesses (e.g., a Class 3 match) which 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 with size (1.27W, 2.71W and 6.51W for the 256K-entry, 512K-entry and 1M-entry devices respectively). The results are analogous for the 200Msps level performance.

5.2 Traffic

As we have mentioned, the concept for longest prefix match in IP-Stash is to iteratively search the set-associative array for progressively shorter prefixes until a match is found. The number of classes and the bit-bounds of those classes determine not only the necessary 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 for the same level of performance.

[7] Results from our modified Cacti that supports high associativity are better than the ones obtained by the multi-bank approximation. We are currently working on convincing ourselves of their validity.

To evaluate the performance of IPStash on real searches, we obtained three traffic traces collected by the NLANR MOAT team [41]. 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 [42]. We selected the routing table snapshot dates to be as close as possible to the capture dates of the traffic traces so we can have realistic simulation results (Table 8).

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

Table 8: 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 11 shows these percentages averaged over the three simulations. Given a specific assignment of prefix lengths to classes we can compute a single coefficient, the average number of accesses per search which characterizes search latency. A Class 1(0) match requires just a single access, a Class 2 match requires one more, while a Class 3 match requires yet another for a total of 3 accesses. For the class definition discussed in Section 2.3 and the distribution of matches in Figure 11 this coefficient is equal to 2.55 accesses/search.

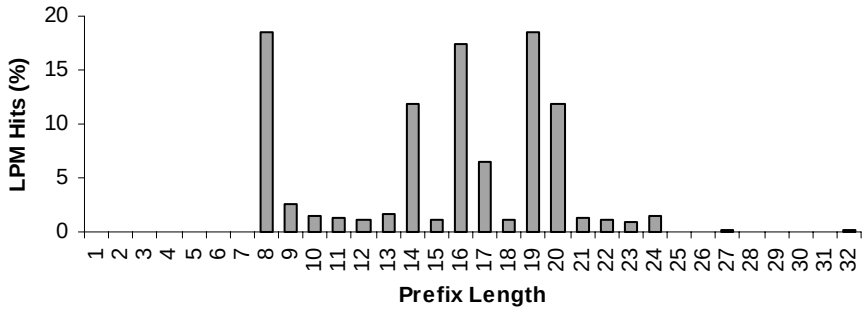

FIGURE 11. Longest-prefix match hits per prefix length (arithmetic average for all tables).

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 the average number of accesses/packet heavily towards high values. We believe that the traffic we examined gives us conservative results (greater than 2 accesses per packet) although at the moment we have no concrete data to back this up. Our theory is that this traffic represents a single outgoing link from a single organization to the outside world. Thus, most of the NASA Ames traffic that is destined for remote sites 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 (Figure 12, left side). We believe that if we had access to the aggregate MAE-West traffic we would see many more packets resolving their addresses 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 (Figure 12, right side).

Despite what we think is unrepresentative traffic, we can change class bounds so we can move to a lower memory access coefficient. If, for example, we change Class 1 to contain bits 19:24 we have an average of 1.9 accesses/packet for the same traffic. This of course affects the size of the expanded routing

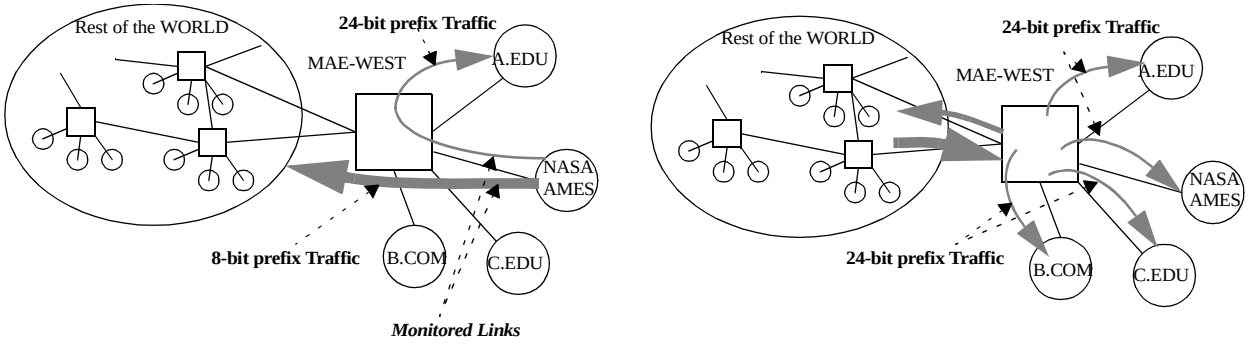

FIGURE 12. A theory why NASA AMES traffic through MAE-West shows so many 8-bit prefix hits. Outgoing traffic to the rest of the world directed mostly by 8-bit prefixes outweighs *local* traffic directed by 24-bit prefixes (left illustration). In contrast, incoming traffic to MAE-West from the outside world should use mostly 24-bit prefixes to reach local entities (MAE-West Incoming Traffic —right illustration).

tables and consequently power consumption. Class definition must balance increased size with a lower accesses/packet average to obtain optimal power consumption for a desired search throughput.

Table 9 shows how the coefficient number of 2.55 accesses/packet obtained from our simulations 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-entry and 512K-entry IPStash devices; we simply extrapolate for the 128K-entry and 1M-entry devices.

IPStash	32-Way	64-Way	128-Way	256-Way
Entries	128K	256K	512K	1M
Bank configuration	32-way x 1	32-way x 2	32-way x 4	32-way x 8
Access Time (ns)	3.04	2.97	3.26	4.8
Cycle Time (ns)	1.38	1.34	1.33	1.69
Average Search latency	3.52	3.42	3.39	4.31
Average Search throughput (pipelined) (Msps)	284	292	295	232
Max Frequency (MHz)	725	746	752	592
Total Energy (all banks) (nJ)	2.15	4.26	9.04	21.71
Power at 170MHz, 66Msps (Watts)	0.37	0.72	1.54	3.69
Power at 255MHz, 100Msps (Watts)	0.55	1.09	2.31	5.54
Power at 300MHz, 118Msps (Watts)	0.65	1.27	2.71	6.51
Power at 600MHz, 236Msps (Watts)	1.29	2.56	5.42	10.86 @ 500MHz (200Msps, 2ns cycle)

Table 9: Cacti's power and timing results for the NASA Ames traffic

Comparing these results with some of the best published industry data (Table 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 work.

	66MSPS	100MSPS
IPSTASH	3.69 W @ 170Mhz,1M-entry,	5.54 W @ 255MHz, 1M-entry
SIBERCORE ULTRA18M (512K-ENTRY)	100Mhz (72bit entries) 6W with full power management 15W at full-associative search	—
MICRON	1.3W per Mbit @ 50 MHz	3.47W per Mbit @ 133 MHz 2.60W per Mbit @ 100 MHz
CYPRESS SEMICONDUCTOR	4.75W per Mbit @ 66MHz	—

Table 10: Overall comparison with TCAMs

6 Related Work

The use of TCAMs for routing table lookup was first proposed by McAuley and Francis [2]. 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 [3]. Kobayashi et al. suggested associating each TCAM entry with a priority [4], 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 [6]. He used a combination of pruning techniques and logic minimization algorithms to reduce the size of TCAM-based routing tables. These algorithms reduce the cost and power consumption of those devices up to 45%. Another method was proposed for multi-field packet classification using TCAMs, where a pre-processing step is used to reduce the size of a TCAM [7].

Zane et al. [8] 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 authors used bit selection algorithms and trie-based algorithms to direct and limit the lookup process in a part of the memory and thus reduce power.

The idea of prefix expansion was initially introduced by Srinivasan and Varghese [11]. 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 [12][13].

Many researchers used caches to speed up the translation of the destination addresses to output port numbers. Results for Internet traffic studies [15][16][18] showed 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. Estrin and Mitzel [9] 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 cache in general purpose processors, such as algorithms proposed in [10] and [13]. 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. These algorithms concentrate on efficient use of cache in general purpose processor instead of designing alternative cache architectures for IP-lookup.

Talbot et al. [17] studied several traces captured from different access routers. A profiling of the address bit frequency determined which bits should be used for cache indexing. Bits whose value has equal probability to be either 1 or 0, are chosen so that cache references are more evenly distributed. 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.

Chiueh et al. [10][16] proposed two schemes to improve performance IP address caching. The first scheme shifts address bits before indexing the cache. Effectively, it uses a fixed length prefix of the IP address as the key, thus some level of address aggregation is achieved. The second scheme utilizes the fact that many routing prefixes share the same destination, thus further aggregation of IP addresses can be achieved by carefully choosing the indexing bits, which need not be contiguous. As a result, a smaller prefix can be generated, which leads to higher aggregation. Both of the proposed approaches need routing table dependent computation, which could decrease the effectiveness as the routing table changes.

Cheung et al. [19] formulated the problem of assigning part of routing table to a different cache hierarchy as a dynamic programming problem, then introduced a placement algorithm to minimize the overall lookup speed. Besson et al. [20] also evaluated hierarchical caches in routers and their effect on IP-lookup. Jain [21] studied cache replacement algorithms for different types of traffic (interactive vs. non-interactive). Pink et al [14] 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 [22] 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 bits 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 machine, 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 the tag. The data side of the IPStash contains two fields: the original unexpanded length of the prefix (so that correct lookup results are guaranteed) and the output port information. IPStash offers unparalleled simplicity compared to all previous proposals while being fast and power-efficient at the same time.

7 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 (sorting & partitioning), power management, 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 (and powered-up selectively) 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 on set-associative memories 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. However, this technique inflates the routing tables (in our case to about twice their original size) which necessitates two things: i) using an IPStash with the same number of bits but about twice the number of entries of a TCAM, and ii) using skewed associativity to maximize the efficiency of the limited number of 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 40% 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.

8 References

[1] C. Labovitz, G.R. Malan and F. Jahanian. Internet Routing Instability. The IEEE/ACM Transactions on Networking, Vol. 6, no. 5, pp. 515-528, 1999.

[2] 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.

[3] D. Shah and P. Gupta. Fast Updating Algorithms for TCAMs. IEEE Micro, 21(1):36-47, January-February 2001.

[4] M. Kobayashi, T. Murase, and A. Kuriyama. A Longest Prefix Match Search Engine for Multi-Gigabit IP Processing. In Proceedings of the International Conference on Communications (ICC 2000), pages 1360-1364, New Orleans, LA, 2000.

[5] A. Seznec and F. Bodin. Skewed Associative Caches. Proceedings of PARLE’93, June 1993.

[6] H. Liu. Routing Table Compaction in Ternary CAM. IEEE Micro, 22(1):58-64, January-February 2002.

[7] J. van Lunteren and A. P. J. Engbersen. Multi-Field Packet Classification Using Ternary CAM. Electronics Letters, 38(1):21-23, 2002.

[8] F. Zane, G. Narlikar, and A. Basu. CoolCAMs: Power-Efficient TCAMs for Forwarding Engines. IEEE INFOCOM, April 2003.

[9] D. Estrin and D. Mitzel. An assessment of state and lookup overhead in routers. IEEE INFOCOM, May 1992.

[10] T.C. Chiueh and P. Pradhan. Cache Memory Design for Network Processors. Proc. High Performance Computer Architecture, pp. 409-418, 1999.

[11] V. Srinivasan and G. Varghese. Fast Address Lookups Using Controlled Prefix Expansion. ACM Transactions on Computer Systems, 17(1):1-40, February 1999.

[12] 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.

[13] B. Lampson, V. Srinivasan and G. Varghese. IP lookups using multiway and multicolumn search. Proceedings of IEEE INFOCOM, vol. 3, pages 1248-56, April 1998.

[14] A. Brodnik, S. Carlsson, M. Degermark and S. Pink. Small Forwarding Tables for Fast Routing Lookups. ACM SIGCOMM, September 1997.

[15] C. Partridge. Locality and route caches. NSF Workshop on Internet Statistics Measurement and Analysis (http://www.caida.org/ISMA/Positions/partridge.html), 1996.
[16] T. Chiueh and P. Pradhan. High performance IP routing table lookup using CPU caching. IEEE INFOCOM, April 1999.
[17] Talbot, Sherwood and Lin. IP caching for Terabit speed routers. Global Communications Conference (Globecom'99), Rio de Janeiro, Brazil, pages 1565-1569, December, 1999.
[18] X. Chen. Effect of caching on routing-table lookup in multimedia environments. IEEE INFOCOM, April 1991.
[19] G. Cheung, and S. McCanne. Optimal Routing Table Design for IP Address Lookups Under Memory Constraints. Proc. INFOCOM, 1999, pp. 1437-44.
[20] E. Besson, and P. Brown. Performance Evaluation of Hierarchical Caching in High-Speed Routers. Proc. Globecom, 1998, pp. 2640-45.
[21] 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.
[22] H. Liu. Routing Prefix Caching in Network Processor Design. IEEE ICCCN2001, October 2001.
[23] Anthony Gallo. Meeting Traffic Demands with Next-Generation Internet Infrastructure. Lightwave, 18(5):118-123, May 2001. Available at http://siliconaccess.com/news/Lightwave_may_01.html
[24] G. Huston. Analyzing the Internet's BGP Routing Table. The Internet Protocol Journal, 4, 2001.
[25] R. Mahajan, D Wetherall, and T. Anderson. Understanding BGP Misconfiguration. In Proceedings of SIGCOMM '02, Pittsburgh, PA, August 2002.
[26] IDT. http://www.idt.com/products
[27] Netlogic microsystems. http:// www.netlogicmicro.com
[28] Micron Technology. http://www.micron.com
[29] Sibercore Technology. http://www.sibercore.com
[30] Intel IXP2850 Network Processor. http://www.intel.com/design/network/products/npfamily/ixp2850.htm
[31] IBM PowerNP Network Processors. http://www.chips.ibm.com/products/wired/network_processors.html
[32] EZ Chip Network Processors. http://ezchip.com/html/in_prod.html
[33] Network and Communications ICs. http://www.agere.com/enterprise_metro_access/network_processors.html
[34] Vitesse IQ2200. http://www.vitesse.com/products/categories.cfm?family_id=5&category_id=16
[35] F. Baboescu, S. Singh, and G. Varghese. Packet Classification for Core Routers: Is there an alternative to CAMs? IEEE INFOCOM, April 2003.
[36] Y. Rekhter, T. Li. An Architecture for IP Address Allocation with CIDR. RFC 1518, Sept. 1993.
[37] 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.
[38] Norman P. Jouppi. Improving Direct-Mapped Cache Performance by the Addition of a Small Fully-Associative Cache and Prefetch Buffers. In Proceedings of the 17th Annual International Symposium on Computer Architecture, pages 364--373.
[39] A. Agarwal, M. Horowitz, J. Hennesy "Cache performance of operating systems and multiprogramming workloads." ACM Transactions on Computer Systems 6, 4 (November 1988).
[40] A. Agarwal and S. D. Pudar, "Column-associative caches: a technique for reducing the miss rate of direct-mapped caches," In Proceedings of the 20th Annual Intl. Symposium on Computer Architecture, May 1993.
[41] Passive Measurement and Analysis project, National Laboratory for Applied Network Research. *http://pma.nlanr.net/PMA*
[42] RIPE Network Coordination Centre. *http://www.ripe.net*