A New IP Lookup Cache for High Performance IP Routers

Guangdeng Liao
University of California, Riverside
Riverside, CA, USA, 92521
gliao@cs.ucr.edu

Heeyeol Yu
University of California, Riverside
Riverside, CA, USA, 92521
hyyu@cs.ucr.edu

Laxmi Bhuyan
University of California, Riverside
Riverside, CA, USA, 92521
bhuyan@cs.ucr.edu

ABSTRACT
IP lookup is in the critical data path in a high speed router. In this paper, we propose a new on-chip IP cache architecture for a high performance IP lookup. We design the IP cache along two important axes: cache indexing and cache replacement policies. First, we study various hash performance and employ 2-Universal hashing for our IP cache. Second, coupled with our cache indexing scheme, we present a progressive cache replacement policy by considering Internet traffic characteristics. Our experiments with IP traces show that our IP cache reduces the miss ratio by 15% and a small 32KB IP cache can achieve as high as 2Tbps routing throughput.

Categories and Subject Descriptors
C.2.6 [**Internetworking**]: Routers.

General Terms
Design, Performance, Measurement.

Keywords
IP Routers, Cache Indexing, Cache Replacement Policies.

1. INTRODUCTION
The demand for high-speed and large-scale routers continues to surge in networking fields. Packet processing for such routers, like IP lookup and packet classification, performs a fast lookup on a large scale rule table. For example, an IP router must perform an IP routing table lookup every time to forward a packet to the next hop.

Since a fast packet forwarding constitutes a router's critical data path, several schemes have been developed based on three major techniques: Ternary Content Addressable Memory (TCAM), trie-based, and hash-based schemes. A TCAM provides a deterministic and high-speed packet lookup [16, 20], but its cost and power dissipation become prohibitively large due to its brute-force search. Unlike TCAM, the trie-based scheme uses a tree-like data structure to successively classify a packet a few bits at a time [4, 8], its drawback, however, lies in its inherent nature of memory space consumption and slow sequential memory accesses. In contrast, hash-based schemes employ a flat data-structure to achieve potentially smaller memory sizes [7, 11, 24, 28]. However, these hash-based schemes ultimately suffer from the setup complexity and several off-chip accesses [7, 24, 28].

Unlike the above schemes which heavily depend on long latency off-chip memory accesses or brute-force searches, a scheme that uses a fast on-chip IP cache can have increased IP lookup throughput without high power dissipation. Authors in [3, 9, 25] present an IP lookup cache architecture since IP traces exhibit strong duplication of destination addresses in the form of temporal locality. By envisioning the benefits of an IP cache, authors in [2, 12] place the IP cache in front of TCAM- or trie-based IP lookup tables to have multifold increase in throughput. However, the existing IP caches are CPU L1/L2 like caches and we observe that the IP cache can be designed much more efficiently considering IP traces characteristics.

The efficiency of IP caches is primarily defined by the cache indexing, which maps an incoming IP address into a cache set. A good cache indexing scheme can uniformly distribute the incoming IP addresses across cache sets in order to reduce cache conflict misses. Existing IP caches simply employ the modular hash, which is used by CPU caches, due to its easy hardware implementation. However, we observe that the modular hash is not a perfect fit for IP caches due to its uneven cache accesses.

In addition to the cache indexing scheme, a cache replacement policy also has a significant performance impact on IP caches. A clear understanding of cache access patterns is important in order to design a cache replacement policy. Authors in [22] study flow level Internet traffic characteristics and show that the distribution of Internet flow popularity follows the Zipf-like distribution. This is also valid even in IP lookup where a flow is defined as a destination IP. Fig.1 shows the IP address popularity distribution of packet frequency and destination IP rank. In the figure, we take Funet and PMA traces from NLANR [17, 19] and plot the number of packets in each destination IP in a reverse order and with a log scale. The slopes of Funet and PMA are -0.92 and -0.93, respectively, indicating that IP address popularity in IP traces follows a Zipf distribution.

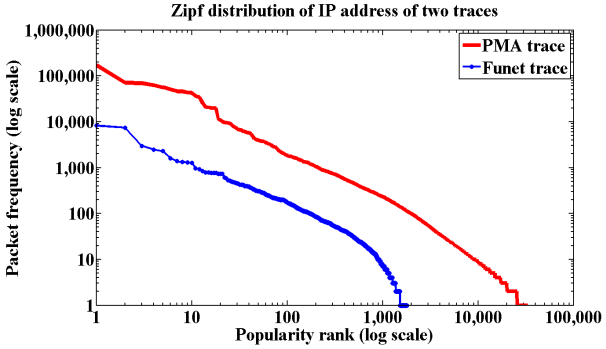

Figure 1. IP address popularity distribution.

The Zipf distribution in IP traces not only confirms IP caching effectiveness due to the strong IP address duplication, but also exhibits skewed flow popularity. Although it cannot be shown in

DAC'10, June 13-18, 2010, Anaheim, California, USA.

Fig.1, we observe from the traces that 14% flows in Funet and 21% flows in PMA are small (or unpopular) flows with only one packet and hence do not exhibit any temporal locality. The existing Least-Recently-Used cache replacement policy (LRU) is unable to fully exploit these characteristics and thus a more efficient cache replacement policy is needed.

In this paper, we carefully design our IP cache along the two important axes: cache indexing and cache replacement policies. Prior studies on CPU caches [10, 14] have demonstrated that XOR-based and Prime-based hash can reduce cache conflict misses for the SPEC CPU benchmark. Motivated by these studies, we extensively study the performance of various hash functions and propose a new cache indexing scheme for IP caches by employing 2-Universal hashing. In hash literature, two Universal hash functions have been analyzed and confirmed that they can lead to a more even distribution of load across hash buckets [6]. Therefore, we split our IP cache into two cache banks, each indexed by a separate Universal hash function. In order to reduce the hashing hardware complexity, we carefully study the bit distribution of IP addresses and select 16 important bits as key bits to a hash function. Using the tailored index bits reduces the hashing hardware complexity by a factor of 2 while retaining the same cache performance.

To further support our proposal, we design a progressive cache replacement policy which aims at keeping popular flows as long as possible and avoiding the eviction of popular flows by unpopular flows. The progressive replacement policy is aware of the Internet traffic characteristic and can be easily implemented over multiple cache banks. In order to evaluate our IP cache design, we develop a trace-driven IP cache simulator and experiment it with IP traces. The experimental results show that our IP cache reduces the miss ratio by 15%, and a 32KB IP cache can achieve on average 2Tbps routing throughput. When our new IP cache is adopted by hash- and TCAM-based lookup schemes, it increases throughput by 1.5X and reduces power by 1.7X compared to the schemes using the existing IP cache, respectively.

The rest of the paper is organized as follows. Sec. 2 provides the IP lookup background knowledge necessary to understand this paper. Sec. 3 elaborates our proposed IP cache while Sec. 4 presents experimental results. Later, we present conclusion in Sec. 5.

2. BACKGROUND AND RELATED WORK

2.1 IP Lookup Schemes

The three representative IP lookup schemes, namely trie, hash, and TCAM have different lookup complexities of *O(W)*, *O(1)*, and 1, respectively, in terms of the clock cycle. Although each scheme has its own pros and cons, a TCAM scheme provides a one-clock IP lookup solution and is preferred by router industries due to its easy-to-implement router design and the worst case lookup support.

It is noted that since TCAMs inherently suffer from a prohibitively large power consumption, researchers propose a hash-based scheme using on-chip Bloom filters [7, 24, 28]. Fig. 2 shows a hash-based IP lookup scheme [7] with five prefixes. Given an IP address, *W* address bits are partitioned and each group of partitioned bits is a hash key group located in an off-chip hash table which provides *O(1)* lookup complexity. Before off-chip hash table accesses, the expensive off-chip access requests are filtered out by using Bloom filters so a memory-efficient approximate membership query is warranted. In our example, we partition *W* bits into groups of 2 bits and there are *W*/2 Bloom filters with each designated to a group. Once a packet with IP address 101011 reaches a network processor (*W*=6), we need two expensive off-chip accesses to the hash tables A and B even though Bloom filter can filter other unnecessary off-chip accesses. Although two hash tables have matched prefixes 10* and 1010*, we select prefix 1010* and its next hop as the packet's next hop because of the longest prefix match in the IP lookup.

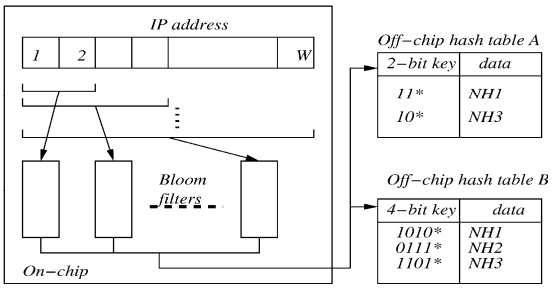

Figure 2. Hash-based IP lookups using Bloom filters and hash tables. An example of 5 prefixes when *W*=6.

Despite the fact that this hash-based IP lookup scheme provides an *O(1)* lookup complexity and better power efficiency when compared to a trie scheme of an *O(W)* lookup complexity and a TCAM scheme of a one-cycle lookup, it inevitably wastes an expensive on-chip memory to filter out unnecessary off-chip accesses, and slow off-chip memory accesses are still needed for an IP lookup. Schemes in [7, 24, 28] typically use 2Mbits on-chip Bloom filters. An IP cache of the same size can provide a fast on-chip IP lookup solution since the IP cache miss ratio is small [9, 25]. Note that if there is a cache miss in an IP lookup, we need to complete the IP lookup by using contemporary off-chip hash-based IP lookup schemes.

2.2 On-chip IP Lookup Cache

2.2.1 IP Lookup Cache

Unlike CPU caches, IP lookup caches maintain the IP lookup history by using IP address bits for cache indexing. Authors in [20] propose a set-associative memory architecture, rather than a cache architecture, by employing a modular indexing scheme. However, such a modular indexing method inevitably suffers from a large collision ratio. Furthermore, since they do not support the 'don't-care' bits, their scheme inflates a routing table and this inflated table does not fit into a small on-chip memory, but the large off-chip main memory. Authors in [9, 25] propose an IP cache by exploiting strong duplication of IP addresses to provide fast IP lookups in network routers. Other TCAM- and trie-based schemes in [2,12] utilize an IP cache scheme to boost up lookup throughput by multi-fold. However, they simply use a modular indexing scheme and LRU, and provide neither an efficient indexing method for a small collision rate, nor Internet-traffic-aware cache replacement. In contrast, our scheme provides a low-collision-rate indexing scheme using two-Universal hashing and a new cache replacement policy considering Internet traffics characteristics.

2.2.2 Cache Indexing

Using alternative cache indexing functions is a popular technique to reduce conflict misses by achieving a more uniform cache access distribution across cache sets [10, 14, 27]. In CPU caches, authors in [10] evaluate the performance of an XOR-based hash for a number of different cache organizations and conclude

that the XOR-based hash is a promising indexing scheme to most cache organizations. A more recent study, performed by Kharbutli [14], shows the XOR-based hash's pathological behavior and proposes a prime-based hash that is resistant to pathological behavior and yet is able to reduce conflict misses for L2 caches. However, the prime-based hash suffers from a cache fragmentation which loses a small fraction of cache sets since a prime number does not fit in power of 2. Also, it is only suitable for L2 caches due to the complex prime hash calculation. Motivated by these studies, our paper extensively studies the performance of various hash functions and reveals that existing alternative CPU cache indexing schemes are not promising enough for IP caches which have different IP address stream behavior. Our result in Sec. 4 shows that our 2-Universal hashing is more promising than any existing hash functions used in CPU caches.

3. NEW IP CACHE ARCHITECTURE

This section presents our new IP lookup cache architecture for high performance routers. We carefully design the IP cache along two axes: cache indexing and cache replacement policies. The detailed cache designs are elaborated in the following subsections.

3.1 IP Cache Organization

Cache indexing maps data into a cache set and has a significant performance impact on caches. Our aim is to make a uniform distribution that can reduce conflict misses. Universal hash functions are well-recognized in hashing literature to generate an even distribution of load over hash buckets and they are relatively easy to implement from a hardware's perspective [21]. We study the performance of various hash functions and observe that 2-Universal hash achieves the best performance (in Sec. 4). Therefore, we split our IP cache into two cache banks, each indexed by a separate Universal hash function.

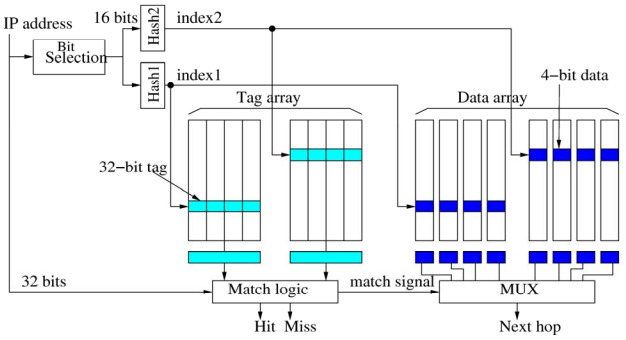

Figure 3. Our IP cache architecture.

Fig. 3 illustrates our IP cache hardware design, where a cache set is addressed by a 32-bit IP address using two Universal hash functions (Hash1 and Hash2). Each cache set in a bank can have 4 cache blocks (or columns in the figure) and each block has a 32-bit IP address served as a tag and a 4-bit next hop. We do a bit-by-bit analysis of IP addresses and select 16 important bits as key bits in order to reduce the Universal hashing hardware complexity. Sec. 3.2 describes the selection process of these particular bits in detail. In order to do an IP lookup, the router processor issues an IP address to the cache. The cache first locates the two cache sets corresponding to the two hashes (Hash1 and Hash2) of the 16 bits and then does the tag check with the IP address in parallel. If the operation is hit in the cache, the next hop is read; otherwise, the hardware replacement unit is triggered to select a cache line for the new IP address.

3.2 Bit Selection and Hash Implementation

The hash function, which we use as an indexing scheme to a cache bank, belongs to the hash function class H_3 and this function is easily H/W-implementable [21]. The detail definition and hardware implementation is the following. Let $A = \{0,1,\ldots,2i-1\}$ and $B = \{0,1,\ldots,2j-1\}$, where i is the number of bits in the key space, i.e. IP address bits, and j is the number of bits in the cache set space. Suppose Q denotes the set of all $i \times j$ boolean matrices. For a given $q \in Q$ and $x \in A$, let $q(k)$ be the k^{th} row of the matrix Q and x_k the k^{th} bit of x. Then, one of H_3 hash functions, $h_q(x): A \to B$ is defined as

$$h_q(x) = x_1 \cdot q(1) \oplus x_2 \cdot q(2) \oplus \cdots \oplus x_i \cdot q(i) \qquad (1)$$

where $\cdot$ and $\oplus$ denote the binary AND operation and the XOR operation, respectively. Fig. 4 shows a circuit implementation example of H_3 class hash function of i=3 and j=2. For a hash function in our IP cache, i=32 since IP address bits are 32, and j=log(# of cache sets). Since hash functions in H_3 are the same except the parameter, each hash function can be configured from a generic chip by providing different parameters.

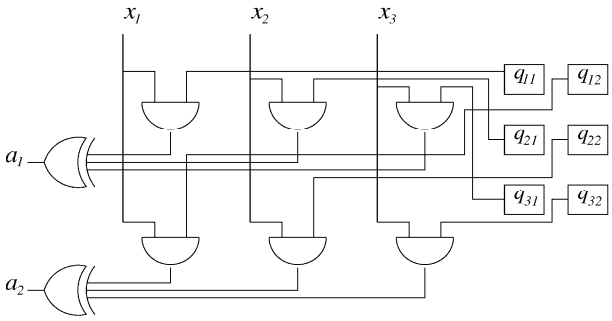

Figure 4. Circuit implementation of a hash function for key=$x_1x_2x_3$, hash matrix elements=q_{ij}, hash index=a_1a_2.

According to the implementation in Fig. 4, hashing latency and hardware complexity increase rapidly with the increase in the key bits. We study a bit distribution of IP addresses with the goal to reduce the number of key bits. We measure the average values of the bits distributed in IP addresses and show them in Fig. 5(a) (the first bit is the MSB). The best key bits (or the important bits) should be those with an average value of 0.5; meaning that they are set 50% of the time over a large series of IP addresses. We notice that bits between the range of 8 to 24 are more important than the rest and thus we choose these 16 bits as our key bits. We ignore 8 least significant bits and compare the miss ratios of n least significant bits as key bits in Fig. 5(b), where all miss ratios are normalized to the miss ratio of 32-bit hash. From the figure, we can claim that our tailored key bits can achieve the same performance as 32-bit hash but with the least hardware complexity.

Our 16-bit hash lowers the hardware complexity, which allows the Universal hash to be feasibly deployed on on-chip caches requiring low hash latency and low power consumption. Our circuit implementation shows that one output bit calculation in 32-bit Universal hash needs one 32-bit XOR logic and 32 AND logics, which correspond to 6 gate delays and 63 CMOS gates (31 gates in the XOR logic and 32 gates for AND logics). In comparison, our 16-bit Universal hash only uses one 16-bit XOR

logic and 16 AND logics for calculating one output bit, and these logics correspond to 5 gate delays and 31 CMOS gates (15 gates in the XOR logic and 16 gates for AND logics). Since each gate only takes 10 picoseconds with Intel 60nm fabrication technology [26], the hashing latency is negligible.

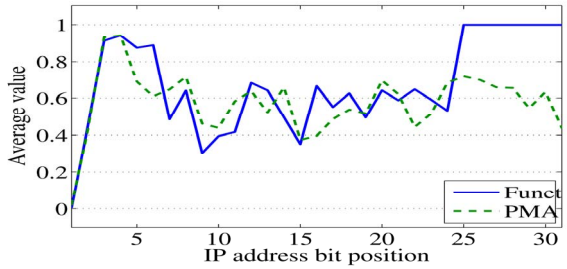

Figure 5(a). Average bit value in IP address.

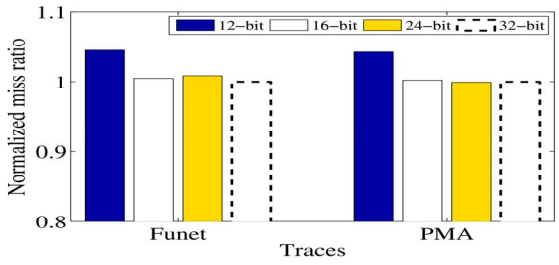

Figure 5(b). Key-bit selection.

3.3 Progressive Cache Replacement Policy

Besides cache indexing, cache replacement policies also have performance impacts on caches [10, 13, 14, 18]. LRU, the default cache replacement policy in CPU caches, has been shown to be promising in the presence of the strong temporal locality. However, LRU is unable to consider the characteristic of highly skewed flow popularity. In addition, it is difficult to implement LRU over multiple cache banks at a reasonable hardware cost. Authors in [10] proposed an affordable LRU hardware implementation by adding a timestamp to each cache line. Once a cache line is accessed, the timestamp is increased by 1 until it is saturated. When a miss occurs, the cache line with the least value of timestamp is replaced. In our experiments we notice that 8 bits for the timestamp are needed for our IP cache in order to achieve comparable cache performance and thus this increases the storage overhead by 22%.

We propose a progressive cache replacement policy for our IP cache by harnessing the IP traces characteristics to address the above issues. Since it is beneficial to store high popular flows in the cache, our policy aims to evict unpopular flows as soon as possible and to keep popular flows as long as possible. We refer to a flow which will not be reused after it is read as a unpopular flow; otherwise, it is categorized as a popular flow. We add one extra bit to each cache line and mark the bit once it is reused. The marked cache line is a popular flow and further accesses to it do not change the bit.

With the support of the extra bit, we count the number of unpopular flows in two corresponding cache sets and use it as a metric when we choose a cache set for the new incoming flow. The cache set with the larger number of unpopular flows is selected; in case of a tie, we choose the left bank for simplicity. Inside each cache set, we replace the cache line in the LRU position by considering the recency, but insert the new flow into the LRU position instead of the MRU position in LRU. Once a cache line is accessed, the policy progressively moves up the cache line by swapping the line with the upper cache line until it reaches up to the MRU position. This design minimizes the time amount that unpopular flows occupy the cache set and thus favors the popular flows. Unlike LRU with 8-bit timestamps, our progressive replacement policy considers both recency and flow's popularity, and only requests one extra bit in each cache line.

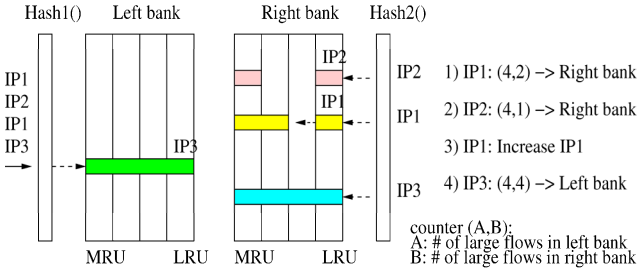

Figure 6. Progressive cache replacement policy.

Fig. 6 illustrates an example of our progressive cache replacement policy. Suppose there are popular flows in the IP cache, which are colored but unlabeled. Given an access sequence of IP1, IP2, IP1, IP3, the policy places IP1, IP2 in the right cache bank according to the number of unpopular flows in cache sets, and inserts them into the LRU position. Once IP1 is read again, IP1 is moved up progressively. Since there are no unpopular flows for IP3, the policy inserts IP3 into the LRU cache line in the left cache bank.

4. EXPERIMENTAL RESULTS

In this section, we start with the study of IP cache performance and then compare with recently published IP lookup schemes: a hash scheme in [24] and a TCAM scheme in [2], in terms of lookup throughput and power consumption. To evaluate our IP cache design, we developed a trace-driven IP cache simulator and experiment it with two traces from [19]. For SRAM and TCAM power modeling, we used tools CACTI [5] and TCAM power estimator [1], respectively. Note that the default replacement policy is LRU and the number of blocks in a cache set is 4, unless stated otherwise.

Table 1. Various cache configurations

Configurations	a	b	c	e	e	f
Indexing	Modular	Prime	XOR	1-Universal	2-Universal	2-Universal
replacement policies	LRU	LRU	LRU	LRU	LRU	progressive replacement

4.1 IP Cache Performance

Since both the cache indexing and the cache replacement policy dominantly determine the cache performance, we conducted experiments with different cache configurations. In our experiments, we include 5 cache indexing schemes (Modular, Prime [14], XOR [10], 1-Universal, 2-Universal) and two cache replacement policies (LRU and our progressive replacement policy). All configurations are shown in Table 1.The existing IP cache [25] is a cache with the modular indexing. Fig. 7 shows IP cache performance comparison by normalizing their miss ratios to the miss ratio of the modular indexing. In CPU caches, the Prime scheme performs better than both the XOR scheme and the Modular scheme [14]. However, the Prime scheme in IP caches

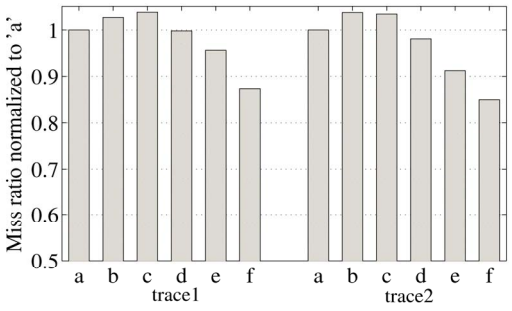

Figure 7. Normalized miss ratio (1K blocks)

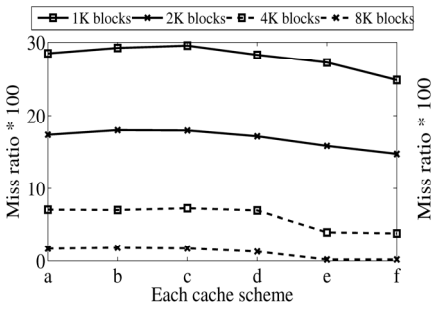

Figure 8(a). Cache miss ratio(trace1)

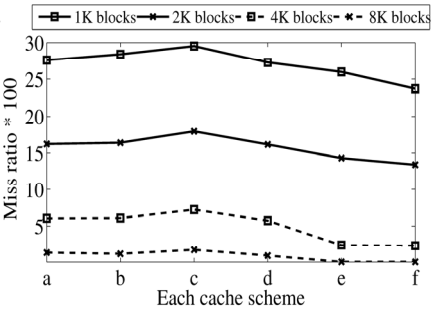

Figure 8(b). Cache miss ratio (trace2)

has larger normalized miss ratios than the Modular scheme. The main reason is that this scheme suffers from the cache fragmentation, which means that several cache sets cannot be utilized due to the prime mod operation. Similarly, the XOR scheme is also unable to improve cache performance by simply using bit-wise XOR operations. Results in Fig. 7 clearly demonstrate that using one Universal hash function ('d') does not reduce cache miss ratios while using two Universal hash functions ('e') with the same cache size reduces miss ratios by up to 9%, compared to the Modular scheme denoted as 'a'. With the combination of our cache indexing, our progressive replacement policy ('f') leverages the traffic characteristics to further reduce miss ratios by 12% and 15% for trace1 and trace2, respectively, compared to the Modular scheme.

In addition, we measured the cache miss ratios of two traces along different cache sizes, from 1K blocks to 8K blocks, as shown in Figs. 8(a) and 8(b). In all sizes, our IP cache (2-Universal plus progressive replacement policy) shows lower cache miss ratios than all other cache schemes. Note that as the cache size increases in power of 2, each cache scheme's miss ratio decreases by half on average. In a case of a 8K cache size, our cache scheme achieves lower miss ratios than 'a' to 'd' cache schemes but shows the similar cache miss ratios to 2-Universal plus LRU ('e'). That is because our 2-Universal indexing can generate more uniform cache access distributions than the first four cache schemes ('a' to 'd'), and our progressive cache replacement policy does not further improve performance, due to very low per-set collision rate which is defined as the number of distinct IP addresses over the number of cache sets.

4.2 Comparison with other IP lookup schemes

In addition to the cache performance study, we compare our IP cache with some other representative IP lookup schemes: hash-based lookup and TCAM schemes, in terms of IP lookup throughput and power consumption. Hash-based IP lookup schemes [7, 24, 28] show an *O(1)* throughput superiority at the on-chip Bloom filter cost. However, if we replace on-chip Bloom filters with our IP lookup cache, we can increase an IP lookup throughput by finishing most lookups with a cache hit. Similarly, if we put our IP cache in front of TCAM schemes, we can significantly reduce TCAM power. Besides our IP cache, we also incorporate other possible IP cache schemes ('a' to 'd') with hash- and TCAM-based schemes for extensive comparison.

1) Lookup throughput comparison with hash-based schemes:

Recently-proposed hash-based IP lookup schemes [7, 24, 28] use on-chip memory for Bloom filters which provide an approximate membership query to improve IP lookup throughput. However, they still need off-chip memory access per lookup. Our IP cache with high cache hit ratios significantly reduces the number of off-chip memory accesses and thus provides higher throughput. In case of a cache miss, we resort to a legacy off-chip hash table to complete an IP lookup.

To measure a lookup throughput, we define average mean access time (AMAT) [18] as follows:

$$AMAT = Hit\ Time + Miss\ Rate \times Miss\ Penalty \qquad (2)$$

where *'Hit Time'* is related to a cache access time if there is a cache hit and *'Miss Penalty'* means off-chip IP lookup time if there is a cache miss. Fig. 9 shows the number of clocks to process an IP lookup under the condition that an on-chip cache access takes 2 clock cycles while an off-chip memory access takes 100 clock cycles. Note that except 'FHT' in [24], all IP cache schemes in *x* axis, are coupled with a legacy hash table. Due to the 0.7% cache miss ratio on average, each IP lookup cache scheme, provides an on-chip speed lookup while 'FHT' with Bloom filters suffers from an off-chip speed lookup. In addition, our IP cache scheme provides on average 1.5 times faster speed than any other cache schemes.

CACTI [5] shows that an on-chip SRAM of 8K blocks has 1.04 *ns* lookup speed in a 90*nm* process technology. If our IP cache scheme processes a packet of the worst-case 20-byte in these speeds, we can achieve 78Gbps ($= \frac{20 * 8\ bits}{2.09 * 1.04 ns}$) for the worst-case lookup. For the average-case lookup, we can assume that a packet has 526B ($= \frac{40 * 40\%}{(40\%+20\%)} + \frac{1500 * 20\%}{(40\%+20\%)}$) since authors in [23] claim that current packet sizes seem mostly bimodal at 40B and 1500B (at 40% and 20% of packets), respectively. Thus, a router with our IP cache scheme can achieve 2Tbps($= \frac{526 * 8 bits}{2.09 * 1.04 ns}$) on average.

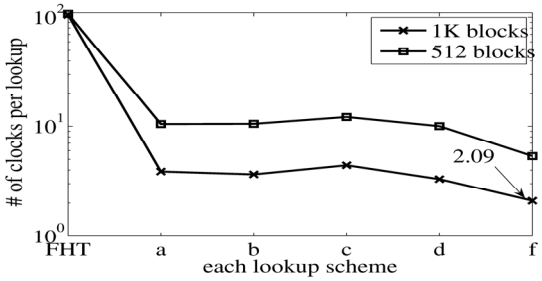

Figure 9. Throughput comparison with hash-based schemes.

2) Power comparison with TCAM-based schemes:

While TCAM can provide the worst-case one-clock lookup, it is criticized as its high power consumption. Placing our IP cache in the front of TCAM reduces TCAM's power consumption while provides the worst-case one-clock lookup with a cache miss. Like

AMAT in Eq. (2), we define average mean lookup power (AMLP) for power comparison as follows:

$$AMLP = Cache\ Power + Miss\ Rate \times TCAM\ Power \quad (3)$$

where *Cache Power* and *TCAM Power* mean lookup powers in a cache and a TCAM, respectively. Fig. 10 shows power consumption of adopting different on-chip IP caches, which is normalized to a TCAM power without a IP cache. Since on-chip SRAMs of 8K blocks and 4K blocks only consume 0.054 and 0.039W [5], respectively, any IP cache can save a TCAM power. Since the TCAM scheme in [2] uses a baseline IP cache, we can consider it as 'a' in the figure. The power comparison shows that adopting an IP cache significantly reduces TCAM power and our IP lookup cache scheme conserves on average 1.7 times power than any other cache schemes.

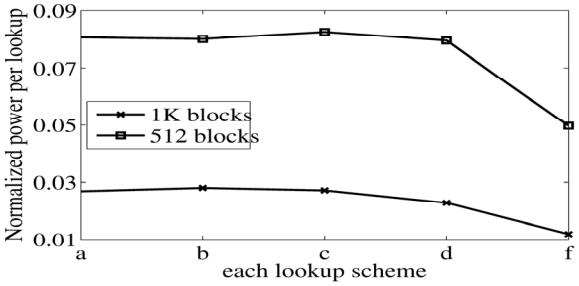

Figure 10. Power comparison with a TCAM-based scheme.

5. CONCLUSION

In this paper, we proposed a new IP cache architecture and designed it along the two important axes: cache indexing and cache replacement policies. To provide high IP cache performance, we extensively study the performance of various hash functions and employ a new Universal hash based cache indexing scheme with two independent cache banks. Some important IP address bits are carefully selected as key bits to reduce the hashing hardware complexity. To further enhance the performance, we design a progressive cache replacement policy by harnessing IP traces characteristics to avoid the eviction of popular flows by unpopular flows. We develop a trace-driven IP cache simulator to evaluate our IP cache design and simulation results show that our IP cache reduces the miss ratio by 15% and a 32KB IP cache can achieve on average 2Tbps routing throughput.

6. ACKNOWLEDGMENTS

The research was supported by NSF grants CCF-0811834 and NEDG-0832108, and a grant from Intel. I would like to thank Yiming Li for her help on this paper.

7. REFERENCES

[1] B. Agrawal, T. Sherwood, Modeling TCAM Power for Next Generation Network Devices, *ISPASS '06.*

[2] M. J. Akhbarizadeh, M. Nourani, R. Panigraphy et al. , A TCAM-Based Parallel Architecture for High-Speed Packet Forwarding, *IEEE Trans. Comput.*, vol. 56, no. 1, 2007.

[3] J. Aweya, IP Router Architectures: An Overview, Technical Report by Nortel Networks, 1999.

[4] A. Basu, G. Narlikar, Fast Incremental Updates for Pipelined Forwarding Engines, *IEEE/ACM Trans. Netw.,* vol. 13, 2005.

[5] CACTI: http://www.hpl.hp.co.uk/personal/NormanJouppi/cacti5.html.

[6] J. Carter, M. Wegman, Universal Classes of Hash Functions, *Journal of Computer and System Sciences*, 1979.

[7] S. Dharmapurikar, P. Krishnamurthy, D. E. Taylor, Longest Prefix Matching using Bloom Filters, *SIGCOMM '03.*

[8] W. Eatherton, G. Varghese, Z. Dittia, Tree bitmap: Hardware /Software IP Lookups with Incremental Updates, *SIGCOMM Comput. Commun. Rev.,* vol. 34, no. 2, 2004.

[9] D. Feldmeier, Improving Gateway Performance with a Routing-Table Cache, *INFOCOM 1988.*

[10] A. Gonzalez, M. Valero, N. Topham et al., Eliminating Cache Conflict Misses through Xor-based Placement Functions, *ICS '97.*

[11] J. Hasan, S. Cadambi, V. Jakkula et al., Chisel: A Storage-Efficient, Collision-free Hash-based Network Processing Architecture, *ISCA '06*.

[12] W. Jiang, Q. Wang, Beyond TCAMs: An SRAM based Parallel Multi-Pipeline Architecture for Terabit IP Lookup, *INFOCOM '08.*

[13] X. Jiang, N. Madan, L. Zhao et al., CHOP: Adaptive Filter-based DRAM Caching for CMP Server Platforms, *HPCA '10*.

[14] M. Kharbutli, Y. Solihin, J. Lee, Eliminating Conflict Misses Using Prime Number-Based Cache Indexing, *IEEE Transactions on Computers*, vol. 54, no. 5, 2005.

[15] S. Kaxiras, G. Keramidas, IPStash: a Power-Efficient Memory Architecture for IP-lookup, *MICRO 36.*

[16] W. Lu, S. Sahni, Lowe Power TCAMs For Very Large Forwarding Tables, *IEEE INFOCOM*, 2008.

[17] S. Nilsson, G. Karlsson, IP-address lookup using LC-tries, *IEEE Journal on Selected Areas in Communication*, vol. 17, no. 6, 1999.

[18] D. A. Patterson, J. L. Hennessy, *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann Publishers Inc., 1990.

[19] Passive Measurement and Analysis Project, National laboratory for Applied network Research (NLANR). [Online]. Available: http://pma.nlanr.net/Traces/Traces.

[20] V. C. Ravikumar, R. N. Mahapatra, L. N. Bhuyan, EaseCAM: An Energy and Storage Efficient TCAM-Based Router Architecture for IP Lookup, *IEEE Trans. Comput.*, vol. 54, no. 5, 2005.

[21] M. V. Ramakrishna, E. Fu, E. Bahcekapili, Efficient Hardware Hashing Functions for High Performance Computers, *IEEE Transactions on Computers*, 1997.

[22] W. Shi, M. H. MacGregor, P. Gburzynski, Load Balancing for Parallel Forwarding, *IEEE/ACM Trans. Netw.*, 2005.

[23] R. Sinha, C. Papadopoulos, J. Heidemann, "Internet packet size distributions: Some observations," USC/Information Sciences Institute, Tech. Rep. ISI-TR-2007-643, May 2007.

[24] H. Song, S. Dharmapurikar, J. Turner, J. Lockwood, Fast Hash Table Lookup using Extended Bloom Filter: An Aid to Network processing, *SIGCOMM '05*.

[25] B. Talbot, T. Sherwood, B. Lin, IP Caching for Terabit Speed Routers, *GLOBECOM '99.*

[26] S. Thompson et al., "130nm Logic Technology Featuring 60nm Transistors, Low-K Dielectrics and Cu Interconnects," Intel, Tech. Rep., May 2002.

[27] H. Vandierendonck, K. D. Bosschere, XOR-based Hash Functions, *IEEE Trans. Comput.*, Vol. 54(7). 2005.

[28] H. Yu, R. Mahapatra, L. Bhuyan, A Hash-based Scalable IP Lookup using Bloom and Fingerprint Filters, *ICNP '09.*