Improving Route Lookup Performance Using Network Processor Cache

Kartik Gopalan and Tzi-cker Chiueh

{kartik,chiueh}@cs.sunysb.edu

Computer Science Department, Stony Brook University, Stony Brook, NY - 11794

ABSTRACT

Earlier research has shown that the route lookup performance of a network processor can be significantly improved by caching ranges of lookup/classification keys rather than individual keys. While the previous work focused specifically on reducing capacity misses, we address two other important aspects - (a) reducing conflict misses and (b) cache consistency during frequent route updates. We propose two techniques to minimize conflict misses that aim to balance the number of cacheable entries mapped to each cache set. They offer different tradeoffs between performance and simplicity while improving the average route lookup time by 76% *and* 45.2% *respectively. To maintain cache consistency during frequent route updates, we propose a selective cache invalidation technique that can limit the degradation in lookup latency to within* 10.2%*. Our results indicate potentially large improvement in lookup performance for network processors used at Internet edge and motivate further research into caching at the Internet core.*

1. INTRODUCTION

A central design issue in network processor architecture is to perform route lookup or classification at wire speed. The routing table lookup problem is to find the longest prefix match[1] to a given packet's destination address from a table of prefix/mask pairs. Earlier efforts[2, 3] have focused on designing compact routing table representations and implementing the route lookup algorithms in hardware. An alternative approach is to apply the time-tested approach of caching to minimize the number of times the lookup algorithms are invoked in the first place. The caching technique remembers the results of previous packet lookups for subsequent reuse. Earlier work[4] had shown that cache performance can be improved by a factor of five over generic IP host-address caching. The work exploited the fact that the number of outcomes for route lookup is bounded by the number of output interfaces regardless of the size of routing table. The earlier technique, called *intelligent host address range cache* (IHARC) performs caching based on ranges of lookup/classification keys, where packets with lookup keys in the same range are forwarded to the same output interface. IHARC reduces the number of *capacity misses* by increasing the range in the IP address space covered by each cache set.

In this paper, we address two important issues that the earlier work did not address satisfactorily. The first issue is that of *conflict misses* that arises when multiple ranges are mapped to the same cache set and a range currently residing in the cache set is not the range to which the address being looked up belongs. Every conflict miss triggers an expensive miss handling mechanism which brings the correct routing entry into the cache through a lookup process. Since the miss handling mechanism is typically an order of magnitude slower than the cache, the route lookup performance of the network processor suffers. We propose two techniques to minimize these conflict misses. Both techniques aim to map ranges across cache sets as uniformly as possible, thus avoiding *hot-spot* cache sets that may result in large number of misses.

The second issue is that of maintaining cache consistency in the face of frequent routing table updates. After an update, some of the cached entries may no longer be valid and hence should not be used in routing decisions. The earlier work proposed a *whole-cache invalidation* approach that repopulates the entire cache from scratch every time there is a routing table update. In this paper we propose a *selective cache invalidation* technique to minimize the performance overhead due to frequent route updates. This technique invalidates only those cache sets whose contents might potentially be affected by the routing table update.

At this point an important question arises as to what extent does the cache miss ratio influence the lookup performance? A simple example can provide an idea. Assume that the cache is designed with high-end SRAM that has a lookup latency of $t_h = 2ns$ in case of a hit. Further assume that the miss handling algorithm can reach a routing decision within $t_m = 40ns$ using tables that are stored in relatively slower SRAMs ($10ns$ access latency). For instance, NART[5] algorithm requires at most three memory accesses to make a routing decision. For a miss ratio m, average lookup latency is given by $(t_m \times m + t_h \times (1 - m))$ Without caching, the average lookup latency would be $40ns$ and

0-7695-1524-X/02 $17.00 (c) 2002 IEEE

the lookup throughput would be 25 million lookups per second (MLPS). Caching with a miss ratio of 10% would reduce the average lookup latency to $5.8ns$ ($2 \times 0.90 + 40 \times 0.10$) and increase the lookup throughput to 172 MLPS. Similarly, a miss ratio of 5% would reduce the average lookup latency to $3.9ns$ and increase the lookup throughput to 256 MLPS. This shows that an intelligent caching mechanism can greatly boost the lookup performance.

For the average lookup times reported in rest of the paper, we use the values of $t_h = 2ns$ for hit latency and $t_m = 40ns$ for miss latency. However note that the miss ratio itself is independent of the memory latencies we assume here. Furthermore, for two different miss ratios m_1 and m_2 (say $m_1 < m_2$), the percentage improvement in average lookup latency is given by

$$\left(1 - \frac{t_m \times m_1 + t_h \times (1 - m_1)}{t_m \times m_2 + t_h \times (1 - m_2)}\right) \times 100$$
$$= \left(1 - \frac{m_1 + \frac{t_h}{t_m} \times (1 - m_1)}{m_2 + \frac{t_h}{t_m} \times (1 - m_2)}\right) \times 100$$

Thus the percentage improvement depends on the ratio t_h/t_m of hit latency to miss latency rather than the absolute values of the latencies. Thus we would obtain the same reported percentage improvements for a much slower cache with $t_m = 10ns$ hit latency and $t_h = 200ns$ miss latency, although its absolute lookup throughput would be lower.

The simulation results in this paper are based on three different packet traces collected from the sole router that connects Taiwan's academic network (about 150 academic institutions) to its ISP in California. The three traces (named Trace 1, Trace 2 and Trace 3) were collected during different hours of a 4-weekday period and consist of 12 million, 34 million and 29 million packets each.* The packet traces collected from the Taiwan router might display higher locality than traces from a true backbone router. Thus, in addition to using a trace-driven simulation methodology, we take the following steps to make our simulation scenario closer to the expected scenario in a true backbone router. First, we intentionally decrease the locality of the three traces by interleaving temporally disjoint sub-traces to form the final trace inputs to our simulators. Second, we choose to use a relatively small cache size to force more capacity misses. Finally, our proposed approach caches address ranges rather than individual IP addresses. The size of an address range could be as high as 2^{21}. Even on Internet backbone routers, the locality at the granularity of address ranges of this size can be expected to be much higher than at the granularity of individual IP address.

The routing table used in our simulations is a combination of the routing tables from the Route Views project † router and the Taiwan router. The Route Views router uses multi-hop BGP peering sessions with several backbone routers at interesting locations over the Internet and constructs a routing table with $150,000$ entries that has details for various networks all around the Internet. The Taiwan router has a smaller routing table of 3000 entries with detailed routing information for different locations within Taiwan's academic network. We combined the two routing tables to obtain a merged routing table that has $153,000$ entries. This merged routing table is most appropriate for use with the packet traces collected from Taiwan router since it captures the routing details both inside Taiwan and in rest of the Internet.

The rest of the paper is organized as follows. In Section 2, we review previous work related to caching in the context of the route lookup problem. In Section 3, we describe the original IHARC architecture proposed in the earlier work[4] that serves as the baseline for our work. In Section 4, we propose two new techniques for conflict miss reduction. In Section 5, we present the selective cache invalidation technique. Section 6 gives a summary of main research results.

2. RELATED WORK

There have been several studies on the locality characteristics of the packet destination addresses that justify the use of caching for lookup/classification. In one of the earliest studies, Feldmeier[6] has shown that the route lookup time can be reduced by 65% with the help of caching. A study of traffic in NSFNET backbone by Claffy[7] showed that caching has significant potential to improve route lookup performance. Similarly, Estrin and Mitzel[8] have shown that there is significant locality in packet streams to justify the use of caching in network processors. A study by Partridge[9] showed that a 5000-entry cache can achieve up to 90% hit ratio. Some commercial routers[10, 11] use caching to improve IP lookup performance. Xu *et al.*[12] have proposed a dynamic set-associative caching scheme that speeds up layer-4 lookup and achieves over 90% hit ratio. The work by Pradhan and Chiueh,[4] which precedes our work, exploits the presence of high locality at the *granularity of address ranges* to improve the layer-3 lookup performance by a factor of five and achieves over 90% hit ratio. To the best

*Other publicly available traces from backbone routers are unsuitable for our cache studies as their real IP addresses are encoded for privacy reasons and it is difficult to obtain un-encoded packet traces from backbone routers in the Internet.

†See http://www.antc.uoregon.edu/route-views for project information and http://moat.nlanr.net/infrastructure.html for routing tables.

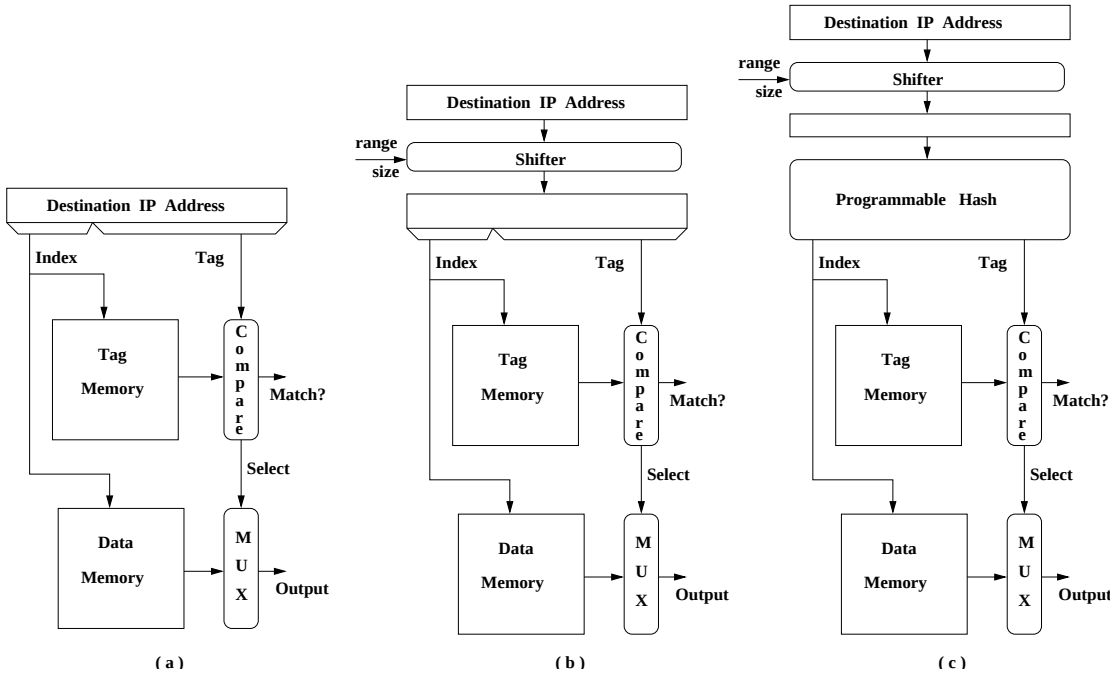

Figure 1. Network processor cache architectures. (a) Host Address Cache (HAC), which is identical to a generic CPU cache, (b) Host Address Range Cache (HARC), which caches host address ranges, and (c) Intelligent Host Address Range Cache (IHARC) in which a programmable hash selects the bits used to index into cache. In HARC and IHARC, the incoming packet's destination address is first right-shifted by a number of bits, which correspond to a contiguous address space range that is mapped to the same output interface.

of our knowledge, till date there have been no published studies that doubt the presence of sufficient locality at the granularity of address ranges.

In other works related to caching in the context of route lookups, Jain[13] has shown that the cache replacement algorithm may need to be different for different traffic types such as interactive and non-interactive traffic. Brodnik *et al.*[14] propose populating the L2 cache of a general purpose processor with a compressed form of the expanded trie representation of routing table in order to reduce lookup time. A significant number of non-caching approaches[15–18] implement lookup algorithms in the hardware itself. In contrast, a caching approach itself is not tied to any specific lookup algorithm.

The selective cache invalidation technique is motivated by the fact that some network environments might encounter very frequent routing table updates as recorded in studies by Labovitz *et al.*[19] and Paxson.[20] Xu *et al.*[12] propose the solution of maintaining a lifetime with each cached entry and invalidating those entries whose lifetime expires. We propose a more pro-active approach that selectively invalidates only those entries that are affected by the cache update and results in better cache consistency.

3. INTELLIGENT HOST ADDRESS RANGE CACHE

The most basic form of network processor cache, known as the *Host Address Cache* (HAC), shown in Figure 1(a), uses the least significant K bits of the destination IP address to index into 2^K cache sets. Since each entry in HAC covers exactly one address, the collective working set of the packet address streams will usually exceed the cache size, especially at backbone routers. The coverage of each cache set can be improved with the *Host Address Range Cache* (HARC) shown in Figure 1(b). The HARC caches address ranges rather than individual IP addresses. Given a routing table, one can statically determine the largest range, say R, that is a factor of the range associated with *every* routing table entry, and leads to the same output port. Then at run time the hardware right-shifts $\log R$ bits of an incoming packet's destination and uses the least significant K bits of the right-shifted value to index into the cache. Thus the coverage of each cache set in HARC is R times the coverage in HAC.

The coverage of each cache set can be further improved by exploiting the fact that the number of possible outcomes of a routing table lookup is equal to the number of output interfaces in the router, which is typically small. By carefully

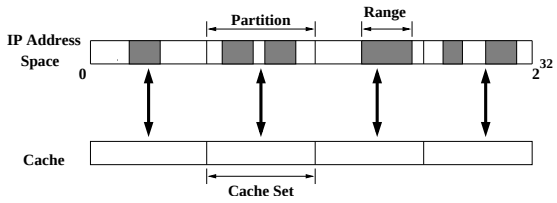

Figure 2. The K index bits divide the IP address space into 2^K partitions. Each partition is mapped to one cache set and is populated by one or more ranges.

choosing the hash function for indexing into the cache, we can merge the address ranges that are *disjoint* in IP address space but share the same output interface. This idea leads to the *Intelligent Host Address Range Cache* (IHARC) shown in Figure 1(c). The IHARC chooses K index bits from the right-shifted input address to index into the data and tag arrays. The K index bits are not necessarily the least (or most) significant K bits but are appropriate bits selected from all the $32 - R$ bits of IP address. As shown in Figure 2, the K *index bits* divide the IP address space into 2^K partitions, each of which is mapped to one cache set. Each partition contains a number of address ranges and each range is associated with an output interface that is different from its neighboring address ranges.

Let us illustrate the idea behind IHARC with the routing table given in Table 1. The routing table maps 16 four-bit addresses to three distinct output interfaces based on eight prefix/mask pairs. Assume the cache has 4 cache sets of size one entry each. Thus the number of partitions is also 4 and the number of index bits used is 2 ($\log 4$). Let the four address bits be labeled as $b3$, $b2$, $b1$, and $b0$ from the most significant to the least significant. Table 1 shows the partition structure that results from using the most significant bits *b3* and *b2* as index bits. There are two ranges (prefix/mask pairs) per partition, and hence per cache set. Table 2 shows the partition structure when bits *b1* and *b3* are selected as index bits in that order. The previously disjoint address ranges 000X/1110 and 010X/1110 in Table 1 which have the same output interface 1 now merge into one contiguous range 00XX/1100. Similar merge occurs with address ranges corresponding to output interfaces 2 and 3, thus reducing the total number of distinct address ranges from 8 to 4. Furthermore, each partition now contains only one address range as opposed to two address ranges in the earlier case. For the same cache size, this gives a 100% hit ratio once the cache is fully populated.

The above example demonstrates the power of a more general *index bit selection* algorithm that identifies the most desirable K index bits in the destination address. Let S be the set of K index bits selected. Set S divides the IP address space into 2^K partitions. Each partition contains a number of distinct address ranges competing for the same

Partition	Prefix/Mask	Output Port
0	000X/1110	1
	001X/1110	2
1	010X/1110	1
	011X/1110	2
2	100X/1110	3
	101X/1110	2
3	110X/1110	3
	111X/1110	2

Table 1. Partition structure that results from using the most significant bits $b3$ and $b2$ as index bits. Bits marked X are ignored during route lookup.

Partition	Prefix/Mask	Output Port
0	00XX/1100	1
1	01XX/1100	3
2	10XX/1100	2
3	11XX/1100	2

Table 2. Partition structure that results from using index bits $b1$ and $b3$. Bits marked X are ignored during route lookup.

cache set at run time. Figure 3 briefly describes a greedy index bit selection algorithm, details of which can be found in the prior work.[4] The algorithm grows the size of the index bit set S from 0 to K. At each step, the algorithm includes a bit j in set S that minimizes $Score(S \cup \{j\})$. The term $Score(S)$ measures the overall desirability of the set of index bits S and is defined as follows.

$$Score(S) = \sum_{i=1}^{2^{|S|}} M_i(S) \; + w \times \sum_{i=1}^{2^{|S|}} |\mathcal{M}(S) - M_i(S)| \quad (1)$$

$M_i(S)$ is the number of ranges in the ith partition resulting from the set of index bits S, and $\mathcal{M}(S)$ is the average of the metric $M_i(S)$ over all partitions i. Each address range corresponds to a cacheable entity. The first term of Equation 1 represents the total number of cacheable entities competing for the entire cache. The second term is called the *deviation term*. It quantifies the deviation in the number of cacheable entities across all the partitions induced by the set of index bits S. In other words, the first and second terms measure the extents of capacity and conflict misses respectively. The weighting factor w in Equation 1 determines the relative importance of conflict miss reduction with respect to capacity miss reduction.

The index bit selection algorithm is invoked whenever a new routing table is installed on the router or whenever the existing routing table has been updated sufficient number of times to render the old index bits ineffective. Although a greedy heuristic, the algorithm is quite effective in practice in reducing the miss ratio by a significant margin. The principal drawback of this algorithm is the complexity in com-

$S = \emptyset$
for $i = 1$ to K
  $min = \infty$
  for each bit j not in S
    if ($Score(S \cup \{j\}) < min$)
      $min = Score(S \cup \{j\})$
      $candidate = j$
  $S = S \cup \{candidate\}$

Figure 3. A greedy index bit selection algorithm to determine the bits used for indexing into cache.

puting $Score(S \cup \{j\})$ for each candidate index bit j over K iterations. For instance, selecting 13 index bits out of possible 32 bits of IP address in a routing table with $153,000$ entries takes about 2 minutes on a Pentium-4 1GHz machine. This implies that expensive index bit re-selection cannot be performed very frequently such as in face of frequent routing table updates.

4. CONFLICT MISS REDUCTION

In this section, we explore two techniques for conflict miss reduction. The first technique ensures that each partition contains a similar number of address ranges. The second technique allocates a variable number of cache sets to each partition based on how many address ranges it contains.

4.1. Balancing the Load among Partitions

If a partition contains large number of ranges, then all those ranges will compete for the partition's associated cache set, thus increasing the probability of conflict miss in that cache set. One good *heuristic* to balance the access "load" among partitions is to minimize the deviation in the number of address ranges in each partition. Prior work[4] accounted for this deviation by including the second term in Equation 1. Since the focus of prior work was to reduce the capacity misses, the significance of the deviation term in reducing the conflict misses was not explored. We improve upon the earlier deviation term in two ways.

First, we note that a *linear* sum of absolute differences $|\mathcal{M}(S) - M_i(S)|$ is not a good metric for reducing the deviation in the number of address ranges since it fails to capture large variations among the absolute differences themselves. For instance, sum of the two absolute differences 1 and 19 (i.e. 20) would be smaller than the sum of the two absolute differences 10 and 11 (i.e, 21). From our point of view, the $(10, 11)$ pair is more desirable than $(1, 19)$, since the former gives more balanced range assignments. This distinction is easily captured by a quadratic term such as *standard deviation*. Thus we redefine Equation 1 as follows:

$$Score(S) = \mathcal{M}(S) + w \times \mathcal{D}(S) \quad (2)$$

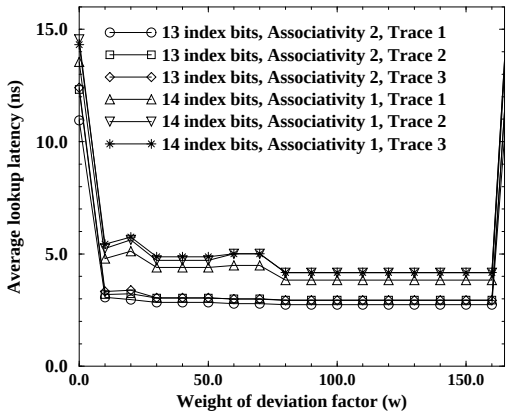

Figure 4. Average lookup latency vs. weight of deviation factor w. The lower three curves correspond to a 2-way associative cache with 2^{13} cache sets. The upper three curves correspond to a 1-way associative cache with 2^{14} cache sets. Beyond $w = 160$, latency increases to around 20ns and is not shown due to considerations of scale. (Hit latency = 2ns and miss handling latency = 40ns).

where $\mathcal{M}(S)$ is the average of the metric $M_i(S)$ over all partitions i and $\mathcal{D}(S)$ is the *standard deviation* in number of cacheable entities across all partitions.

$$\mathcal{M}(S) = \frac{\sum_{i=1}^{2^{|S|}} M_i(S)}{2^{|S|}} \quad (3)$$

$$\mathcal{D}(S) = \left(\frac{\sum_{i=1}^{2^{|S|}} (\mathcal{M}(S) - M_i(S))^2}{2^{|S|}} \right)^{\frac{1}{2}} \quad (4)$$

Secondly, prior work assumed a value of 1 for the weighting factor w and did not explore the possible advantages of tuning this factor. We show that selecting an appropriate value of w can significantly reduce the level of conflict misses in the cache. A small w implies that the extent of capacity miss reduction is more important in determining a bit's desirability as an index bit and a large w implies that the extent of conflict miss reduction is more important. The challenge is to find the right value of w that strikes a balance between conflict and capacity misses and achieves the lowest overall cache miss ratio.

We performed a series of cache simulations to study the effect of varying w. Figure 4 plots the average lookup latencies obtained from using index bits selected via Equation 2 for different values of w. We examine two different cache structures, namely, a 2-way associative cache with 2^{13} cache sets and a 1-way associative cache with 2^{14} cache sets. Note that both caches have the same effective capacity of 2^{14} entries. For each curve in the figure, the smallest latency is obtained when the value of w is between 80 and

Index bits : Assoc.	Trace type	Lowest miss %	Avg. lookup latency (ns)
14:1	Trace 1	4.84	3.84
	Trace 2	5.70	4.17
	Trace 3	5.70	4.17
13:2	Trace 1	1.94	2.73
	Trace 2	2.50	2.95
	Trace 3	2.50	2.95

Table 3. Smallest miss ratios and average lookup latencies for different simulation scenarios. Trace 1 produces a smaller latency since fewer number of packets in Trace 1 result in lesser interleaving and consequently higher locality.

160. Beyond $w = 160$, the selected index bits cause heavy fragmentation of IP address space into very small ranges and the average lookup latencies increase steeply due to increase in capacity misses. Another fact to note is that 2-way associative cache gives lower miss ratio (and hence smaller latency) than the 1-way associative cache with the same effective capacity. This behavior is similar to traditional CPU caches. Table 3 summarizes Figure 4 by listing the smallest miss ratios and latencies obtained for different simulation scenarios. In the rest of the paper, we use a 2-way associative cache with 2^{13} cache sets for the performance evaluations. Also, we present only those results obtained from Trace 2 since this trace has the worst locality characteristics and results from the other two traces follow similar trends.

When 13 index bits are selected via Equation 1, a two-way associative cache with 2^{13} cache sets yields the smallest lookup latency of 2.98ns (2.60% miss ratio) with Trace 1 and 3.21ns (3.19% miss ratio) each with Traces 2 and 3. This shows that Equation 2 is a better measure for selecting index bits since it yields smaller lookup latencies.

To further understand how the lookup latency varies with w, we examine three factors - a) the maximum number of ranges in any partition, b) the average deviation in the number of ranges per partition and c) the total number of ranges across all partitions. Table 4 shows that as w increases, the maximum number of ranges in any partition initially decreases steeply and then stabilizes when w is between 80 and 160. This trend implies that the level of conflict miss is reduced rapidly with an initial increase in w and is responsible for the initial dramatic drop in the cache miss ratio shown in Figure 4. The standard deviation also decreases initially and becomes stable after $w = 80$ which demonstrates that the number of ranges in different partitions become more balanced as w is increased. The total number of address ranges across all partitions increases with w due to address range fragmentation, indicating that the level of capacity miss is traded for a reduction in the level of conflict

Weight (w)	Max. # of ranges	Standard deviation	Total # of ranges ($\times 1000$)
0	862	80.7	155
10	190	14.8	301
20	183	14.7	322
40	133	12.1	425
60	160	11.9	437
80	142	10.1	677
100	142	10.1	677
160	142	10.1	677
200	3014	10.0	1006

Table 4. Range distribution statistics as different set of index bits are selected by varying the weight of deviation factor w. (Cache sets = 2^{13}, associativity=2).

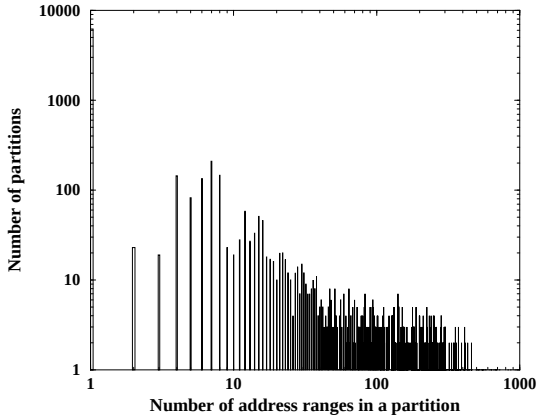

Figure 5. Number of partitions (Y-axis) that contain a given number of address ranges (X-axis) in log-log scale. The 13 most significant bits of IP address are chosen as index bits. Around 6000 partitions contain exactly one address range. The maximum number of ranges in any partition is 3053.

miss. Note that the ideal value range for w that we derive for this case, i.e. between 80 and 100, is specific to the routing table and packet traces used in our simulations. A different routing table and/or trace would require a tuning of the weight w using the same methodology as described above.

4.2. Variable Cache Sets Mapping

The first technique involved balancing the level of conflict misses across partitions by selecting appropriate index bits. While it yields a significant reduction in miss ratio, it also involves an effort in tuning the weight w and executing an expensive index bit selection algorithm. In this section, we propose an alternative conflict miss reduction scheme that is less expensive in terms of computation overhead and, unlike the first technique, it does not trade the level of capacity miss for conflict miss reduction.

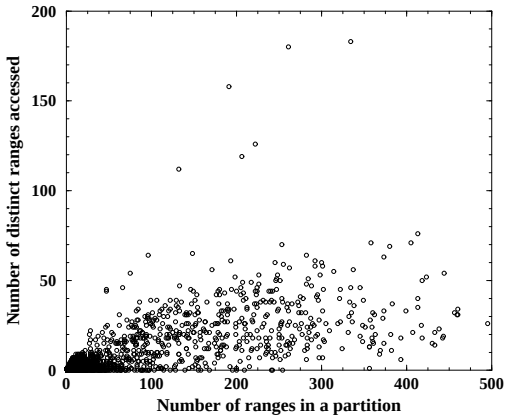

Figure 6. Distribution of number of distinct ranges accessed (Y-axis) in partitions with a given number of address ranges (X-axis). Index bits are the 13 most significant bits of IP address. Partitions with more than 500 ranges are not shown due to consideration of scale.

We have assumed up until now that the 2^K partitions, generated by K index bits are mapped to exactly one cache set each. By relaxing the mapping between partitions and cache sets, one can allocate a variable number of cache sets to each partition. Consider figure 5 which shows the distribution of number of address ranges across 8192 partitions in the merged routing table. Although a large majority of partitions have only one range, a significant number of others have a large number of ranges and, in the extreme case, one partition has 3053 ranges. Figure 6 shows that partitions that have a larger total number of ranges are more likely to cause higher level of conflict miss, since more number of distinct ranges get accessed in such partitions. In order to reduce conflict misses, some highly populated partitions can be *split* over multiple cache sets while others that are less populated can be *merged* into common cache sets. We call the process of determining how many cache sets to assign to each partition as the *variable cache sets mapping* (VCSM).

In general, we select the K most significant bits of IP address as index bits to define the 2^K partitions. The next N most significant bits are selected as additional *splitting bits* that can split each partition into at most 2^N sub-partitions. As shown in Figure 7, the VCSM approach maps the $2^{(K+N)}$ sub-partitions to 2^K cache set lines according to a *programmable mapping logic* which follows a generic K-bit decoder. The additional flexibility provided by N splitting bits allows each of the 2^K decoder outputs to be expanded into 1, 2, 4 and up to 2^N cache set lines, or alternatively, to be mapped to a common cache set line along with other decoder outputs. We will represent each possible

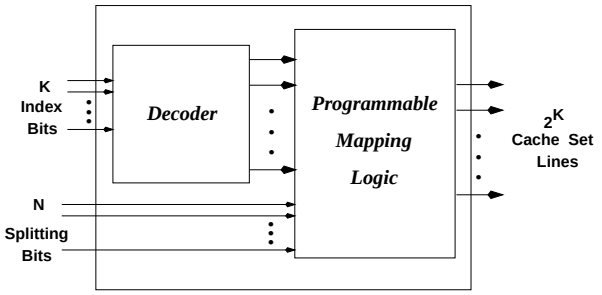

Figure 7. Programmable logic maps the decoded results of K index bits and N splitting bits to 2^K cache set lines.

cache configuration by the pair (K, N). Note that a K-bit decoder is part of every generic cache and doesn't increase the complexity of the cache design. The only additional hardware complexity due to programmable mapping logic is proportional to the number of splitting bits N and can be implemented with low overheads using state-of-the-art programmable logic device (PLD) technology.

The mapping logic is statically derived as follows before being programmed into the cache. The 2^K partitions of IP address space are first sorted in an increasing order according to the number of address ranges they contain. Then the partitions are mapped to cache sets depending on their positions in the sorted order. For example, in a "**16-4**" mapping logic, the first x partitions in the sorted order are mapped into common cache sets in groups of 16 partitions and the remaining y partitions are split over 4 cache sets each. In order to determine x and y, we solve the pair of equations $x+y = 2^K$ and $x/16+4y = 2^K$ to obtain $x = 48\times 2^K/63$ and $y = 15 \times 2^K/63$. Similarly, we can define other kinds of mapping logic such as "**2-2**", "**4-4**", "**16-8**" and so on.

Table 5 shows the cache simulation results for $(13, N)$ cache, i.e, $K = 13$ index bits and different values of splitting bits N. From the $N = 0$ case, we see that the worst case average lookup latency is 12.32ns (or 81 million lookups per second (MLPS)) when VCSM is not used. However, in the $N = 3$ case, the latency can be improved to 6.75ns (or 148 MLPS) with the cache configuration $(13, 3)$, i.e., an improvement of 45.2%. Another noticeable trend in Table 5 is that, as the number of splitting bits increases from 0 to 3, the miss ratio decreases. This is expected since more splitting bits provide the flexibility of splitting highly populated partitions into smaller pieces (provided that there are enough cache sets to accommodate the split), thus reducing the level of conflict miss. However, for 4 splitting bits, there are not enough cache sets to accommodate the split of sufficient number of partitions such that the miss ratio could be further reduced.

The average lookup latency could be improved to less than 5ns if we increase the associativity of cache from 2 to

Splitting bits (N)	Mapping logic	Standard deviation of ranges	Miss ratio (%)	Average lookup latency (ns)	Improvement (%)
0	No VCSM	73.32	27.16	12.32	Base case
1	2-2	56.56	22.19	10.43	15.3
2	4-4	34.97	17.14	8.51	30.9
3	1024-8	25.58	12.52	6.75	45.2
4	32-16	30.75	14.96	7.68	37.7

Table 5. Miss ratios and average lookup latencies for a cache of size 2^{13} sets and associativity of 2. The most significant 13 bits are chosen as the index bits and Trace 2 is used. Each row lists the mapping logic that produced the smallest miss ratio. (Hit latency = 2ns and miss handling latency = 40ns).

4 and use 12 index bits while keeping the total number of cache entries the same at 2^{14}. The disadvantage of increasing associativity is that it comes at the expense of increasing the complexity of cache design.

If we use the VCSM approach in conjunction with the load balancing technique (described in Section 4.1), the lookup latency can be reduced at most from 2.95ns for a $(13, 0)$ cache to 2.81ns for a $(13, 3)$ cache. i.e., a mere 4.7% improvement. The reason for small improvement is that once the index bit selection process is completed, the deviation in number of address ranges is already so small that VCSM scheme does not have much scope to balance the address ranges any further.

To summarize, without any form of optimization and using 13 most significant bits of IP address as index bits, we obtain an average lookup latency of 12.32ns. By applying load balancing technique alone, the latency can be reduced by 76% to 2.95ns. Using the VCSM approach alone, the latency can be reduced by 45.2% to 6.75ns. In spite of smaller percentage improvement than load balancing technique, the VCSM approach offers two advantages. First it provides lower computational overhead by avoiding the expensive index bit computation. Secondly, unlike the load balancing technique, it does not increase the level of capacity misses in order to reduce conflict misses.

5. SELECTIVE CACHE INVALIDATION

As mentioned in Section 1, frequent routing table updates can have an adverse impact on performance of network processor cache. To maintain network processor cache consistency, earlier work[4] had proposed a naive method of invalidating the entire cache upon every routing table update and re-populating the cache from scratch. Such a *whole-cache invalidation* scheme can effectively wipe out any performance gains from the caching technique.

An alternative approach is *selective invalidation* that invalidates only those cache sets that are actually affected by the routing update. A routing table entry with a P-bit network address prefix could cross 2^m partitions, if the number of index bits that are not in the most significant P bits of the IP address is m. When this routing table entry is updated, a selective invalidation approach invalidates the cache sets corresponding to these 2^m partitions. More concretely, if the number of index bits is K, then those cache sets that have the same remaining $K - m$ index bit values as in the updated P-bit address prefix are to be invalidated.

Let's examine the effect of routing table update on cache miss ratio using the two invalidation techniques. In our simulations, we periodically add, delete or update one routing table entry. New entries that are added to the routing table consist of randomly generated `<prefix, mask, output port>` triples, The routing table entry chosen as a target of each delete or update operation is picked randomly from the set of most recently accessed routing table entries. This procedure ensures that, when using selective invalidation, there is a high probability that each routing table update indeed renders certain cache sets invalid, thus resulting in a larger number of subsequent cache misses. During simulations, an average of 8 to 9 cache sets were invalidated due to each update.

Figure 8 compares the miss ratio from the whole-cache invalidation against that from the selective cache invalidation at different routing table update frequencies. In the absence of routing table updates, one can obtain an average lookup latency of 2.95ns (or 339 MLPS). At the frequency of one update every $10,000$ packet lookups, the average lookup latency using whole-cache invalidation technique degrades to 8.5ns (or 118 MLPS), i.e. a degradation of 240%. On the other hand, using selective invalidation technique, the average lookup latency degrades to only 2.98ns (or 336 MLPS), i.e. a degradation of only 10.2%.

An important concern with selective cache invalidation is that with each routing table update, the current set of index bits no longer remains the most ideal choice and would *po-*

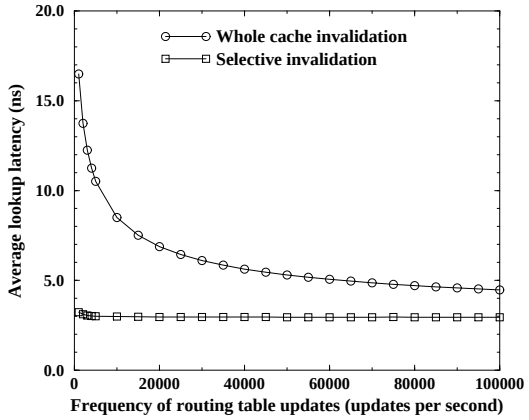

Figure 8. Average lookup latency vs. routing table update frequency for a cache of size 2^{13} and associativity=2 when $w = 100$ and Trace 2 is used.

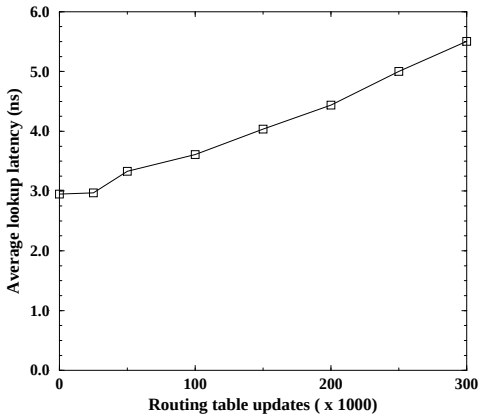

Figure 9. Average lookup latency vs. number of updates to routing table. Each update consists of either an add, delete or an output port update operation.

tentially need to be recomputed. Since the re-computation of index bits is expensive, we choose *not* to recompute the index bits with every routing table update. Instead index bits could be recomputed at a lower frequency of, say, once every few hours. The price for this decision is that, between re-computations we continue to use the old set of index bits that progressively become less of a good choice after each routing update. Figure 9 shows the performance degradation when we continue using old index bits as the number of updates to the routing table increases from 0 to $300,000$. Correspondingly, the average lookup latency increases from 2.95ns to 5.5ns. Up to $25,000$ updates, the latency increases to only 2.96ns. If the routes are updated even at the rate of 5 updates per second, then $25,000$ updates correspond to 1.4 hours, which is long enough to allow one to re-compute the index bits without adversely affecting the hit ratio. Realistically, we do not expect $25,000$ updates to occur even in one day and hence the index bit re-computation can be performed at a lower frequency of several hours.

The key reason why selective cache invalidation in response to route updates is feasible is that the index bits used in packet lookup cache are usually chosen from the most significant bits of the destination IP address. By definition, since the address prefix of each routing table entry also starts with the most significant bits of the IP address, the number of affected partitions and thus affected cache sets is relatively small. As mentioned above, on the average 8 to 9 cache sets were invalidated as a result of each routing table update. This small number of invalidations not only minimizes the impact on subsequent cache performance but also makes it feasible to perform invalidations within a few cycles. In contrast, the index bits of standard CPU cache are typically the less significant bits of memory addresses. Therefore, it is more difficult to implement a similar selective invalidation mechanism for the standard CPU cache when a page is updated.

6. CONCLUSION

In this paper we addressed two important issues affecting the routing table cache performance that the earlier work[4] did not explore, namely reducing conflict misses and minimizing the penalty due to frequent routing table updates. In order to reduce conflict misses, we propose two techniques that distribute access load evenly across different cache sets. The first technique carefully selects index bits by balancing the number of IP address ranges mapped to each cache set. However, it trades capacity misses for a reduction in conflict misses. The second technique assigns variable number of cache sets to different IP address partitions without executing expensive index bit selection or trading capacity misses. Simulations indicate that the two techniques can reduce the average lookup latency by up to 76% and 45.2% respectively.

In order to reduce the performance penalty due to frequent routing table updates, we propose a selective cache invalidation approach. This approach invalidates only those cache sets that are possibly affected by the routing table update. Compared to a whole-cache invalidation approach, our approach reduces penalty on lookup latency from 240% to 10.2% when the updates are as frequent as once every $10,000$ packet lookups.

We expect that the results obtained from our simulations will definitely hold for major Internet edge routers and hope

that our work will motivate further research into the dynamics of caching in network processors for true backbone routers. We are interested in extending the current techniques to the IPv6 routing table lookup problem and the multi-field packet classification. The challenge here is to keep the complexity of index bit selection algorithm low as the number of bits over which the selection needs to be done increases. Although our current heuristic provides acceptable performance when index bit selection is done infrequently, we believe there is further scope for improvement.

Acknowledgement : We would like to thank Prashant Pradhan for providing the original implementation of the basic IHARC simulator and also Srikant Sharma, Pradipta De and anonymous refrees for providing insightful comments that greatly improved the presentation of this paper.

REFERENCES

1. W. Doeringer, G. Karjoth, and M. Nassehi, "Routing on longest matching prefixes," *IEEE/ACM Transactions on Networking*, **4(1)**, pp. 86–97, Feb. 1996.
2. M. Waldvogel, G. Varghese, J. Turner, and B. Plattner, "Scalable high speed IP routing lookups," in *Proc. of ACM SIGCOMM'97, Cannes, France*, pp. 25–36, Sept. 1997.
3. V. Srinivasan and G. Varghese, "Faster IP lookups using controlled prefix expansion," in *Proc. of SIGMETRICS'98, Madison, WI, USA*, pp. 1–10, May 1998.
4. T. Chiueh and P. Pradhan, "Cache memory design for network processors," in *Proc. of 6th Intl. Symp. on High-Performance Computer Architecture, Toulouse, France*, Jan. 2000.
5. T. Chiueh and P. Pradhan, "High performance IP routing table lookup using CPU caching," in *Proc. of IEEE INFOCOM'99, New York, NY, USA*, pp. 1421–1428, April 1999.
6. D. Feldmeier, "Improving gateway performance with a routing-table cache," in *Proc. of IEEE INFOCOM'88, New Orleans, LA, USA*, pp. 27–31, March 1988.
7. K. Claffy, *Internet Traffic Characterization*, Ph.D. Thesis, University of California at San Diego, CA, USA, June 1994.
8. D. Estrin and D. Mitzel, "An assessment of state and lookup overhead in routers," in *Proc. of IEEE INFOCOM'92, Florence, Italy*, pp. 2332–2342, May 1992.
9. C. Partridge, *Locality and Route Caches*, Position Statement in NSF Workshop on Internet Statistics Measurement and Analysis, San Diego, CA, USA, Feb. 1996.
10. P. Newman, G. Minshall, and L. Huston, "IP switching and gigabit routers," *IEEE Communications Magazine* , Jan. 1997.
11. C. Partridge et al., "A 50-Gb/s router," *IEEE/ACM Transactions on networking* **6(3)**, pp. 237–248, June 1998.
12. J. Xu, M. Singhal, and J. Degroat, "A novel cache architecture to support layer-four packet classification at memory access speeds," in *Proc. of INFOCOM 2000, Tel Aviv, Israel*, pp. 1445–1454, March 2000.
13. R. Jain, "Characteristics of destination address locality in computer networks: a comparison of caching schemes," *Computer Networks and ISDN Systems*, **18(4)**, pp. 243–254, May 1990.
14. A. Brodnik, S. Carlsson, M. Degermark, and S. Pink, "Small forwarding tables for fast routing lookups," in *Proc. of ACM SIGCOMM'97, Cannes, France*, pp. 3–14, Sept. 1997.
15. P. Gupta, S. Lin, and N. McKeown, "Routing lookups in hardware at memory access speeds," in *Proc. of IEEE INFOCOM'98, San Francisco, CA, USA*, pp. 1240–1247, April 1998.
16. Music Semiconductors, *MUAC Routing CoProcessor (RCP) Family, http://www.music-ic.com*.
17. D. Shah and P. Gupta, "Fast incremental updates on Ternary-CAMs for routing lookups and packet classification," in *Proc. of Hot Interconnects-8, Stanford, CA, USA*, Aug. 2000.
18. A. Prakash and A. Aziz, "OC-3072 packet classification using BDDs and pipelined SRAMs," in *Proc. of Hot Interconnects-9, Stanford, CA, USA*, Aug. 2001.
19. C. Labovitz, G. Malan, and F. Jahanian, "Internet routing instability," in *Proc. of ACM SIGCOMM'97, Cannes, France*, pp. 115–126, October 1997.
20. V. Paxson, "End-to-end routing behavior in the Internet," *IEEE/ACM Transactions on Networking*, **5(5)**, pp. 601–615, Oct. 1997.