

# Efficient Prefix Cache for Network Processors

Mohammad J. Akhbarizadeh and Mehrdad Nourani

Center for Integrated Circuits & Systems  
The University of Texas at Dallas  
Richardson, TX 75083  
{eazadeh,nourani}@utdallas.edu

**Abstract**—Conventional routing cache systems store destination IP addresses in their cache directory. We present a routing cache technique that stores the most recently used route prefixes, instead of IP addresses, to achieve significantly smaller cache size. A nesting prefix is partially represented in this cache by its minimal expansions. Such expanded prefixes are obtained using an incremental technique without any modifications to the routing table. Consequently, our cache works with most of the common route lookup algorithms and efficiently maintains coherency with the routing table. Experiments show that for a hit ratio over 0.96 our design can achieve more than 33 times reduction in cache size, compared to a conventional routing cache.

## I. INTRODUCTION

Search is among the most time consuming tasks of network processors, especially the searches corresponding to route lookup (also called *the longest prefix matching* problem or LPM) and packet classification jobs in Internet edge routers. Despite the availability of many hardware search engines, the tendency to use trie-based software algorithms is still strong. Complex trie-based algorithms that try to reduce the number of memory accesses per lookup usually compromise update time and/or memory utilization [1]. Network processor producers prefer simpler and more flexible methods. For example IBM's PowerNP uses a hash table plus a set of Patricia trees each rooted in one location of the hash table for its IP route lookup engine [2]. One way to reduce the average search time is to use routing cache [3]. Similar to the conventional CPU cache systems, routing cache is a fast and relatively small memory that stores the most frequently used data and reduces the average memory access time by making the common cases fast.

Unlike CPU cache that associates memory address to memory data, each entry of a basic routing cache associates a destination IP address with a next hop address (or egress port number). IP addresses do not have any spatial locality of references. Therefore, routing caches only exploit the temporal locality of references within IP addresses. Conventional routing caches are therefore rather big in size and face some resentment by the network processor architects [4]. However, a closer look at the IP addresses shows that the only part of an address used for making the routing decision is the network Id part. Two different destination addresses with the same network Id part but different host Id parts would still take their packets to the same next hop. So, it is conceptually possible to reduce the routing cache size significantly by only

caching the network Id parts of IP addresses. That way all the addresses with the same network Id can be represented by just one entry in the routing cache. In IPv4 CIDR (classless inter-domain routing), the network Ids are stored in the routing table in form of route prefixes. A route prefix is a sequence of bits from left to right followed by consequent *don't-care* bits. A prefix can have any length between 8 and 32. An example is the 18 bits prefix 100000010110111001\* (where \* represents all 14 right hand don't-care bits), also shown as 129.110.64/18.

### A. Background

Based on the above observation, the approach to improve route caching is that upon a LPM search in the routing table, the longest matched prefix is mirrored in a prefix cache. All the future packets that would match the same prefix in the routing table will find their route in the prefix cache and will not need to refer to the routing table. Chiueh and Pradham were the first to introduce this idea in [5]. They also noticed that the parent prefixes make prefix caching more complex than it appears. Briefly, a parent prefix is one that nests another longer prefix. The former is a prefix of the latter. For example, 129.110/16 is a parent of 129.110.128/17. An IP address that matches the longer prefix (child) also matches the shorter (parent). The LPM algorithm, however, must always choose the most specific (longest) matching prefix. For example, 129.110.150.1 matches both 129.110/16 and 129.110.128/17 but must be routed to the destination pointed to be the latter prefix, because it is longer. If a parent prefix is mirrored in the prefix cache, an IP address that matches it might as well have a longer match in the routing table. So, if 129.110/16 is placed into the prefix cache while 129.110.128/17 is in the routing table, 129.110.150.1 that matches the former in the prefix cache would be forwarded to the wrong path unless it was also searched in the routing table. Therefore, there is no point in mirroring parent prefixes in the prefix cache because the routing table has to be looked up anyways. Any prefix in the prefix cache must be disjoint.

Authors in [5] addressed this issue by proposing to expand the entire routing table into disjoint ranges. The outcome of their expansion is a set of range prefixes that do not nest each other. Their *Host Address Range Cache* (HARC) architecture caches such disjoint ranges. Although the exact size reduction with respect to conventional routing cache is case dependent, the paper reports an estimated reduction by a factor of 32. The main concern involving the approach taken by this paper is its

restrictive structure. In other words, for the routing table to be kept as a set of disjoint ranges it may have to be restructured upon each table update. This fact makes table management quite complicated.

### B. Contribution

we report a successful attempt to improve HARC [5] by directly and efficiently caching the routing prefixes, without any special table preprocessing requirement. Our proposed *Reverse Routing Cache* (RRC) improves three aspects of the state of the art. First, mostly original route prefixes are placed into the cache directly. No range manipulation of the routing table contents is needed. Second, RRC does not require any specific organization or transformation of the routing table. Therefore, it can work with just about any route lookup method. Especially, it does not impose a data structure on the routing table that requires reconstruction upon route update. Third, RRC improves the cache storage efficiency. The effective cache size reduction for different RRC sizes will be demonstrated through accurate simulation.

Throughout this paper we sometimes refer to the searchable piece of data as *key*. A key is a destination IP address in case of IP route lookup but could be another sort of data in prefix or range search operations in a classification engine. While this paper focuses on efficient use of routing cache for IP route lookup, we would like to comment that many of the techniques developed here can also be employed with conjunction to some packet classification algorithms. In all the models presented in this paper, to maintain consistency, the routing table is assumed to have a trie-based structure. Among other possibilities this can be a binary-trie (see Figure 1 for an example) or a level compressed trie [1]. To find the longest matching prefix of a key in any of these two structures, the key is used to trace out a path in the trie, remembering the last matched prefix along the path. When there are no more branches to take, the search ends and the longest matching prefix is the last match remembered. The examples in the next section are given with a basic binary trie for simplicity.

## II. PREFIX CACHING

### A. RRC Structure

The methods introduced in this paper are not restricted to any particular cache structure. However, we did all our experimentations with a cache simulator that modeled RRC as a fully associative cache. The spatial locality does not exist among routing table entries. Therefore, cache block size in all the models is one and no prefetching is employed. A benefit of using the fully associative model was that we avoided the effect of set size on the performance comparisons and evaluations.

Since RRC stores prefixes, it is best to implement it as a Ternary CAM module [6], [7]. The cache size is going to be very small. So, TCAM implementation of RRC is quite affordable.

Conventional TCAM-based routing tables sort their contents based on prefix length and use a priority encoder to choose

TABLE I  
RESULTS FOR SAMPLES OF ROUTING TABLES: (A) AADS, (B) MAE-WEST, (C) PACBELL

(a) AADS Site

Sampling Date	Table Size	Number of Parents	Parent Percentage
03/14/02	21649	1453	6.7%
03/15/02	21604	1468	6.8%
03/19/02	21744	1483	6.8%

(b) MAE-WEST Site

Sampling Date	Table Size	Number of Parents	Parent Percentage
03/14/02	18619	1209	6.5%
03/15/02	18681	1211	6.5%
03/19/02	18768	1232	6.5%

(c) PacBell Site

Sampling Date	Table Size	Number of Parents	Parents Percentage
03/14/02	5222	379	7.2%
03/15/02	5239	379	7.2%
03/19/02	3875	233	6.0%

the longest matching prefix among multiple matches. Since parent prefixes are not allowed in cache, each RRC lookup generates no more than one match per search, which means the contents of RRC need not be sorted, unlike conventional TCAM-based routing tables. A new entry can be added to any available location in RRC, which makes RRC updates possible in only one cycle. This also eliminates the need for a priority encoder at the output of RRC, which can reduce the RRC search latencies up to 50% compared to conventional TCAMs of equal size. The authors have confirmed this empirically, though the details in this regard are beyond the scope of this paper. For details about priority encoder design see [8]. On the other hand, note that a TCAM cell employs more number of transistors than a CAM cell that would be used as traditional fully associative cache. A static TCAM requires 16 transistors per cell while a static CAM cell can have only 9 transistors [7].

For the purpose of comparison, a conventional routing cache that stores IP addresses was similarly modeled. This model is referred to as *IP cache* in the rest of this manuscript.

### B. Handling Parent Prefixes

Parent prefixes cannot be directly placed in a routing cache. The reason was explained in Section I. We tried two different approaches to handle parent prefixes in RRC. Each approach is explained separately below.

- **RRC-PR:** Parent prefixes are scarce. Our investigations show that only less than 8% of the prefixes in any major routing table are parent prefixes. To show the scarceness of parent prefixes, studies of a few samples from three main Network Access Points (NAP) are presented in Table I. The three NAPs studied were AADS, MAE-WEST, and PacBell.

Samples were taken on three days: March 14, 2002, March 15, 2002, and March 19, 2002 from the IPMA project website [9]. For example, the third row of Table I(a) shows that on March 19, 2002 number of entries in the sample routing table was 21744 out of which 1483 prefixes were parents that made up 6.8% of the entries in the table. In this example 20261 prefixes, or 93.2%, are disjoint and can be directly mirrored in RRC. As the data in Table I induce, parent prefixes are always less than 8% of the total entries of the routing tables we have analyzed so far.

Assuming that their average popularity is as low as their population, parent prefixes could be totally neglected by RRC. In this strategy, only the disjoint prefixes, that do not nest any other prefix, are allowed to be mirrored in RRC. Section IV shows the detailed simulation results of RRC modeled based on this strategy. We shall refer to this model as *RRC with Parent Restriction* or RRC-PR, hereafter.

■ **RRC-ME:** For many Internet service providers it might not be acceptable to let a traffic flow that corresponds to a parent prefix always go through the slower path, only because the parent prefix cannot be mirrored in RRC. Also, despite being scarce, the parent prefixes might be more popular than others, because they relatively cover a wider portion of the network. For these two reasons, we may need to handle parent prefixes in RRC more intelligently. To address this concern we developed a technique in which a parent prefix is partially represented in RRC with the the shortest expanded disjoint child that matches the given search key. Such prefix is called the *Minimal Expansion Prefix* (MEP).

When looking up a given key, suppose the deepest that the key can traverse in the trie is  $i$  bits and the last remembered prefix node is a parent P (i.e LPM is a parent prefix). Then MEP is created by taking the string of bits traversed so far and appending to it the next bit of the key. The resulting  $i + 1$  bit prefix is the shortest disjoint child of the matching parent that also matches the given key. It can be used to partially represent P in RRC and all the future IP addresses that share the same  $i + 1$  leftmost bits as the given key will hit it. We call this process *minimal expansion*.

Note that this process does not modify the routing table. The obtained prefix should only be added to RRC. Also, note that the minimal expansion technique could be adopted for almost any other routing table structure. The only important requirement is that the parent prefixes must be recognizable.

In the worst case, the minimal expansion process needs  $O(W)$  cycles to finish, where  $W$  is the maximum width of route prefixes. This is when a binary-trie is used for the routing table and the system requires an additional search for minimal expansion. In most cases the additional search is not necessary and minimal expansion requires no extra cycles.

Figure 1 shows an example and how this technique works. For a fictitious 5 bit address space, the given  $key = 10110$  generates a miss in RRC and is looked up in the main routing table. The search stops at the intermediate node  $n$  ( $n = 101*$  and  $i = 2$ ) because branching to right is not possible. The last

remembered prefix (the longest match) is  $p = 10*$ . Since  $p$  is a parent prefix, the minimal expansion technique adds the bit at position 1 of the key to  $n$  to get  $p' = 1011*$ .  $p'$  is added to RRC to partially represent  $p$ . Over time, minimal expansion may occur to the same parent prefix multiple times. So, a parent prefix may be represented in RRC by multiple minimal expansion children. Compared to the total prefix expansion employed by HARC [5], this technique advocates gradual, adaptive, and dynamic expansion of only parent prefixes. RRC with minimal expansion capability is referred to as RRC-ME in the rest of this manuscript.

In its worst case, the process of minimal expansion requires a new search, which means the miss penalty in case of matching a parent prefix is more than the miss penalty in case of matching a disjoint prefix, though this is not the case for most of the systems. Many practical systems can produce MEP at the same time that they produce the search result. However, the authors would like to include an study of the absolute worst case too. The higher miss penalty for parent prefixes increases the overall miss penalty and consequently, the average search time.

If the average search time is  $T_{srch}$ , the cache access time upon hit is 1 time unit, the miss penalty is  $T$ , and the miss ratio is  $m$  ( $m = 1 - h$ , where  $h$  is the hit ratio) then the average search time for IP Cache or RRC-PR is obtained from the equation:  $T_{srch} = (1 - m) + mT$ . In case of RRC-ME, if the miss penalty for parents is  $K_p$  ( $K_p \geq 1$ ) times higher than that of disjoint prefixes and the ratio of misses that end up matching a parent to the total misses is  $\alpha_p$  ( $0 \leq \alpha_p \leq 1$ ) then the average search time is:

$$T_{srch-ME} = (1 - m) + (1 - \alpha_p + \alpha_p K_p) m T \quad (1)$$

For a general purpose microprocessor that employs a software-based LPM algorithm in the external memory, the additional search means doubling the number of memory accesses which almost doubles the miss penalty. Hence, in this case  $K_p = 2$ . On the other hand, for a network processor with an external search engine co-processor, the additional search required for minimal expansion (if it is required at all) is done inside the search engine and all the bus transactions to update RRC-ME are done only once. So, in this case the overhead of minimal expansion process is negligible and  $K_p \approx 1$ , which means  $T_{srch-ME} = T_{srch}$ . We conclude that in most practical cases  $1 \leq K_p \leq 2$ , and it is expected to be equal to 1 in the majority of the systems.

### C. Updating RRC

A Semi-LRU (least recently used) method that conceptually uses a descending age counter for each RRC entry was implemented. The counter range is equal to the size of the cache and is set to maximum at the placement time and at any subsequent matches. If RRC is full and every virtual counter is at a value bigger than zero then the replacement is failed.

1) *RRC Coherence Tasks:* There are two circumstances in which RRC might lose coherence with the routing table: 1)



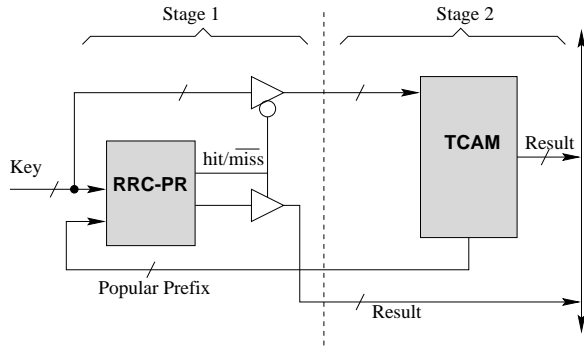


Fig. 2. RRC added to large TCAM to reduce average power consumption.

experiments show. In the absolute majority of cases deletion of a parent prefix causes removal of less than a handful of entries from RRC-ME. Moreover, because parent prefixes constitute a tiny minority in the routing table, the average number of RRC-ME removal operations needed in effect of a table deletion is always very close to one. To assess this empirically, we counted the number of RRC-ME prefixes with a parent in the routing table, at the end of a simulation run. The maximum number of child prefixes in RRC-ME associated to the same parent was 17, which means the maximum number of cycles that it takes to synchronize this RRC-ME's contents when a prefix is deleted from the routing table will be 17. The average number of child prefixes in RRC-ME per parent prefix in the same simulation was 2.8 (including MEPs and others), and 31% of RRC-ME was occupied by these child prefixes. That means the average number of cycles per coherence task when all routing table entries with a mirror in RRC-ME are deleted sequentially is  $\frac{1}{(0.31/2.8)+(1-0.31)} = 1.25$ .

### III. OTHER PRACTICAL ASPECTS OF RRC

■ **Use with network processors:** First and foremost, RRC makes an efficient level one caching subsystem for network processors to significantly reduce the bus transactions with the external routing or classification engine. For example, if the cache read time is 1 unit and the miss penalty is 120, a RRC of size 128 can bring down the average time consumed per search to 9.56 units (see Table II).

■ **Hardware flexibility:** In addition to reducing the cache size, RRC allows more complicated implementations compared to conventional cache designs. Many techniques become affordable only when the cache size is small enough. For example, full associativity or the LRU replacement strategy might not be practical for a 4K IP Cache, especially for a level one cache system. However, an LRU method and the full associativity become viable design choices for a small 128 line RRC.

■ **TCAM power reduction:** Our methodology provides the possibility of implementing RRC for reducing the average power consumption of CAM and TCAM search engines. Even though TCAM is the ultimate choice for high performance routing or classification engines, its high power consumption

is always seen by designers as a negative factor against their advantages[6]. Also, many researchers, e.g. authors in [10], state that the power consumption of Internet is a major concern and urge for solutions to reduce the average power usage both on transmission channels and in the switching/routing points of the Internet. Because TCAM power consumption is linearly proportional to its size and because RRC is a very small TCAM module, by augmenting the big TCAM routing table with a small RRC that captures most of the search requests we can achieve a significant reduction in average power consumption. For example, a 26K TCAM that allocates the sample routing table of Section IV consumes less than 18% of its original dynamic power when accompanied with a RRC of size 64, as our simulations show. Figure 2 shows the block diagram of such system. Note that in this case the RRC is not placed inside the network processor. Rather it is added to the TCAM chip. Clearly, in this application the search speed is not an issue and only the average hit ratio matters. So, parent prefixes can be totally ignored from being mirrored and RRC-PR can be employed. The authors are currently combining RRC with the partitioned TCAM architectures [11] to reduce the average power consumption to a negligible level while providing guarantees for an acceptable peak power consumption. We hope to report the results in near future.

### IV. RRC PERFORMANCE EVALUATION

Based on the methods explained in Section II we developed models for both flavors of RRC: RRC-PR and RRC-ME. We evaluated both models with different cache sizes in the range of 16 to 2048. The packet trace used for these simulations is the same used in [5], collected from the main router of a national laboratory. The routing table used for this experiment was taken from the IPMA project website [9] and contained 26,786 entries. We also simulated IP Cache in a similar architecture, i.e. fully associative with the same semi-LRU replacement strategy. This model was evaluated with the same packet trace and routing tables for various cache sizes from 64 to 26,000.

Figure 3 demonstrates the outcome of our simulations. The horizontal axis of this graph shows the cache sizes on logarithmic scale. As the graph shows, RRC-ME reaches a saturation level of 0.998 when its size exceeds 2048. RRC-PR has a value of 0.956 at that size. For IP Cache the hit ratio is only 0.917 when its size is 2048 and it reaches the hit ratio of 0.999 only when its size reaches 26000, which is almost the size of the routing table itself. Two typical hit ratios are highlighted on this curve. To get an average hit ratio of 0.928, the dotted horizontal line shows that the required RRC-ME size is 128 while the size of IP Cache has to be 3500. This means RRC-ME needs 27.3 times less memory. The second highlighted hit ratio is 0.962 for which the required RRC-ME size is 256 but IP Cache size is almost 8600. This translates to 33.6 times memory reduction for RRC-ME.

Table II shows the miss ratios ( $m = 1 - h$ ), the relative parent miss ratio ( $\alpha_p$ ), and the average search times for the same two

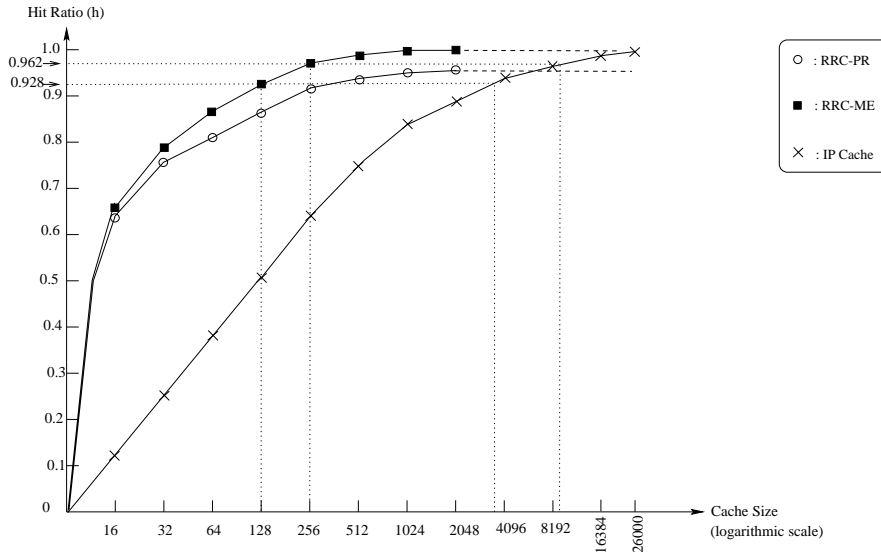


Fig. 3. Hit ratio for various sizes of RRC-ME, RRC-PR, and IP Cache.

TABLE II

AVERAGE SEARCH TIME FOR TWO DIFFERENT RRC-ME SIZES.

RRC-ME size	Miss ratio (m)	Parent miss ratio ( $\alpha_p$ )	$T_{srch-ME}$ [time unit]	
			$K_p = 1$	$K_p = 2$
128	0.072	0.167	9.56	11.00
256	0.038	0.201	5.52	6.44

RRC-ME sizes that were highlighted in Figure 3. Average search times (see Equation 1) are reported for two different systems, one with the minimum  $K_p$  value of 1, the other with the maximum  $K_p$  value of 2. Values of  $\alpha_p$  are obtained through simulation and show the fraction of the missed lookups that match a parent prefix in the routing table. The table shows the effects of cache size as well as  $K_p$ , for two RRC-ME designs providing typical hit ratios. Doubling the RRC-ME size reduces the average search time by 42% and 41% for  $K_p = 1$  and  $K_p = 2$ , respectively. The average search times are reported for a miss penalty of 120 time units and a the cache access time of 1.

## V. CONCLUSION

We proposed a novel prefix caching method, called the *Reverse Routing Cache* (RRC). The RRC methodology and its performance when applied to a trie-based packet forwarding engine are demonstrated. The RRC unit is more than 33 times smaller than a traditional routing cache while achieving the same hit ratio. Unlike the previous art, RRC does not require reconstruction or modification of the routing table. RRC makes an efficient level-one cache for network processors to significantly reduce bus transaction with external routing or classification engines.

## ACKNOWLEDGMENTS

The authors acknowledge Professor Tzi-cker Chiueh and his research group in the State University of New York at Stony

Brook for providing IP packet traces. We would also like to extend our appreciation to Rina Panigrahy and Samar Sharma from Cisco Systems, Inc. for their fruitful discussions.

## REFERENCES

- [1] M. Ruiz-Sanchez, E. Biersack and W. Dabbous, "Survey and Taxonomy of IP Address Lookup Algorithms," *IEEE Network Magazine*, March 2001.
- [2] M. Peyravian, G. Davis, and J. Calvignac, "Search Engine Implications for Network Processors Efficiency," *IEEE Network*, July-August 2003.
- [3] T. Chiueh and P. Pradham, "High Performance IP Routing Table Lookup Using CPU Caching," *IEEE INFOCOM99*, New York, 1999.
- [4] D. Comer, *Network System Design Using Network Processors*, Pearson Prentice Hall, 2004.
- [5] T. Chiueh and P. Pradham, "Cache Memory Design for Internet Processors," *IEEE Micro*, January-February 2000.
- [6] P. Lekkas, *Network Processors Architectures, Protocols, and Platforms*, McGraw-Hill, 2004.
- [7] V. Srinivasan, B. Nataraj, and S. Khanna, "Methods For Longest Prefix Matching in a Content Addressable Memory," *US Patent no. 6,237,061*, January 1999.
- [8] J. Wang and C. Huang, "High-Speed and Low-Power CMOS Priority Encoders," *IEEE Journal of Solid State Circuits*, vol. 35, no. 10, Oct. 2000.
- [9] <http://www.merit.edu>, "Internet Performance Measurement and Analysis Project," 2002.
- [10] M. Gupta and S. Singh, "Greening of the Internet," *ACM SIGCOMM03*, Karlsruhe, Germany, 2003.
- [11] R. Panigrahy and S. Sharma, "Reducing TCAM Power Consumption and Increasing Throughput," *IEEE HotI02*, Stanford University, 2002.