Greedy Prefix Cache for IP Routing Lookups

Zhuo Huang   Gang Liu   Jih-Kwon Peir
Department of Computer and Information Science and Engineering
University of Florida
Gainesville, FL, USA
{zhuang, galu, peir}@cise.ufl.edu

Abstract— As the rapid growth of Internet and the communication link speed, it becomes increasingly challenging for network processors to timely route the incoming traffic to the destination ports. The traditional approach must look up the routing table based on the destination IP address to determine the output port. The ternary CAM approach provides fast associative look up, but is very costly for large routing tables. The trie-based algorithm allows inexpensive searching, but may not satisfy the growing speed requirement. Previous studies showed that the overall routing time can be shortened by adding a small prefix cache for the general trie-based routing algorithms. In caching the prefix, however, the nested prefixes are difficult to cache due to the constraint of the longest prefix matching requirement. This paper presents a greedy prefix caching technique to improve the prefix cache performance that allows caching the largest sub-tree of each prefix including the parent prefixes. Our experiment results show that the prefix cache using the proposed upgrade scheme can reduce the miss ratio by about 6-8% comparing to the best existing prefix caching mechanism.

Keywords—IP lookup, Prefix Matching, Prefix Cache

I. INTRODUCTION

To cope with the intensive growth of Internet, faster and faster fiber links are deployed to the market place. For an Internet edge router, the most difficult and time-consuming task is to timely route the ever-increasing packets to their next hops. The routing path based on the package’s destination IP address is normally recorded in the routing table in each router. As the routing table size grows rapidly along with the high traffic rate, the table lookup becomes the bottleneck for making the routing decisions.

Since a destination subnet with a unique network ID usually consists of a large number of hosts, the packages to these hosts are often routed through the same set of intermediate hops. Instead of recording the <IP-address, Out-port> pair, the routing table size can be greatly reduced by only recording the output port for each routing prefix, denoted by <Prefix, Out-port>. The prefix is a bit string followed by a *, where the * is a wildcard that matches any bit string from the remaining destination IP address. To match of the routing prefix, however, introduces the longest prefix matching (LPM) problem. Instead of finding an exact match, the router must find the longest match of the prefix. Figure 1 (a) shows an example of a routing table with two prefixes. A package with an IP destination address <0100...> matches both prefixes. Because of the LPM requirement, the package will be routed to Port B.

The LPM problem needs to be resolved efficiently by either hardware or software. There are three main categories of LPM approaches including the Ternary Content Addressable Memory (TCAM) [19,2], the trie-based searches [23,5,6,20,15,16,11,17,14] and the hash-based lookups [4,21,10,8,22,12,13]. TCAMs are custom devices that can search all the prefixes stored in the table simultaneously. They incur low delays, but require expensive tables and comparators and generate high power dissipations. Hashed-based approaches use hash tables to store the prefixes. It is power-efficient and is able to handle large prefix tables. The main difficulties of hash-based approaches include collisions of the hash entries, the support for LPM, and the handling of prefix updates. Trie-based approaches use a tree-like structure to store the prefixes with the matched output ports. They consume low power, less storage spaces, and can handle prefix updates easily. In addition, the LPM requirement can be satisfied naturally through searching the trie. The main problem of trie-based approaches is the long lookup latency involving multiple memory operations especially for IPv6 where the prefix lengths can vary from 16 to 64 bits. Figure 1 (b) illustrates an example of lookup an address <0100...> in a binary trie, where the dark node indicates a matching prefix. The first match along the search path is found at <01*→Port A>. The
search continues and finds the longest match at <0100*→Port B>. The <01*→Port A> is referred as the parent or nested prefix of the leaf prefix <0100*→Port B>.

Based on the locality study, previous works [8,5,7] added a small, fast cache to reduce the latency. The recent routing results are saved in the cache, so when such routes are revisited in the near future, the destination ports can be retrieved directly from the cache. The straightforward approach is to simply cache the IP address but it is quite inefficient. Caching the prefix, however, is complicated due to the LPM requirement. Consider that a package with the IP address <0101*...> needs to be routed through the routing table in Figure 1. The matched parent prefix <01*→Port A> cannot be cached since it will route incorrectly for any future package with a destination address <0100*...>.

There have been a number of studies [23,15,1] for caching the prefixes other than the IP addresses. One approach is to expand the entire routing table into disjoint ranges to avoid any nested prefixes [23]. Another recent approach [1] is to abandon caching any nested parent prefix. Instead, they cache the largest sub-tree below the parent prefix, referred as the minimal expansion prefix (MEP). The MEP will not be nested with any other prefix and thus can be cached without encountering incorrect routing.

In this paper, we proposed a greedy prefix cache for caching parent prefixes based on the fact that the parent prefix can be cached as long as all of its children prefixes have already located in the cache. Our study shows that a parent prefix can cover several expansion prefixes, hence can expand the routing table coverage in a fixed size cache. We describe an upgrade technique that can identify cached MEPs and upgrade them to the parent prefix once the corresponding children prefix enters the cache. Our simulation results based on the 24-hour trace from MAWI [18] and the as1221 routing table [3] shows that about 6-8% improvement of cache miss rate can be achieved by upgrading the existing MEPs to the parent prefixes.

The remaining of the paper is organized as follows. Section 2 provides the motivation for caching the parent prefixes. Section 3 introduces the greedy cache with prefix upgrade. This is followed by the performance evaluation of the proposed greedy prefix cache in section 4. Section 5 describes the previous prefix caching approaches. A brief conclusion is given in Section 6.

II. MOTIVATION FOR CACHING PARENT PREFIXES

A prefix lookup on a trie structure starts from the root. At each level, the corresponding IP address bit determines the search path to the left link or to the right link as shown in Figure 2 (a). For simplicity, we consider a single-bit trie, where the dark nodes house a possible matching prefix. Whenever a prefix is matched along the search path, the corresponding output port is recorded. The search continues until it reaches a leaf node. The last recorded output port is selected for routing the package. Since a longer prefix is always matched later, the trie-based algorithm guarantees to find the longest match.

In caching the prefixes, care must be taken when the longest match is a parent prefix. Instead of caching the parent prefix, the minimal expansion prefix (MEP) can be cached as proposed in [1]. A MEP contains the biggest sub-tree within the parent sub-tree where the longest match was found without creating any conflict with the children prefixes. Figure 2 (b) shows the two MEPs marked by the gray color, <011*→Port A> and <0101*→Port A> of the parent prefix <01*→Port A>. These two MEPs can be cached without violating the longest prefix matching requirement. However, the parent prefix <01*→Port A> can be cached to cover both MEPs as long as the child prefix <0100*→Port B> is also located in cache. The longest match will correctly select the Port B for any matched child prefix. All others that match parent prefix will be routed to Port A as shown in Figure 2 (c).

To understand the benefit of caching the parent prefix, we collect the fundamental routing information based on the 24-hour trace from MAWI [18] and the as1221 routing table [3]. We first examine the prefix reference distribution with respect to the leaf, the parent, and the root prefixes as shown in Table 1. The parent prefixes have one or more child prefix and the default root prefix "*" matches any IP address with the entire prefixes as its children. The static distribution provides the number of prefixes in the routing table. The dynamic distribution, on the other hand, is the number of matched prefixes from the real trace. It is interesting to observe that although the parent prefix represents less than

\[ \text{Table 1. The Static and Dynamic Distribution of Prefixes} \]

<table>
<thead>
<tr>
<th></th>
<th>Leaf</th>
<th>Parent</th>
<th>Root</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Static</td>
<td>200615(93.1%)</td>
<td>14835(6.9%)</td>
<td>1(0.00%)</td>
<td>215451</td>
</tr>
<tr>
<td>Dynamic References</td>
<td>5.65M(24.0%)</td>
<td>8.98M(38.2%)</td>
<td>8.87M(37.8%)</td>
<td>23.52M</td>
</tr>
</tbody>
</table>
7% in the routing table, the total references to the parent prefixes from the real-trace are close to 40%. Therefore, caching the parent prefix plays an important role for improving prefix cache performance.

We also collected the average number of MEPs for each parent prefix as shown in Table 2. In this table, the parent prefix is categorized by the average number of children prefixes. A child of a parent prefix can be either a leaf prefix or another nested parent prefix. For example, a parent prefix X with two children can have one child Y which is a parent of a leaf prefix Z. The prefix Y’s MEPs are accumulated into the total MEPs for one-child parent. Importantly however, the prefix Y’s MEPs will be excluded from counting the MEPs for the parent prefix X. In other words, the MEPs will not be double-counted for all the parents. The results indicate that caching the parent prefix is much more efficient than caching the MEPs since it requires several MEPs to cover a parent prefix.

Given the fact that caching the parent prefix is much more efficient than caching the MEP, we further compare the reuse distance using the MEPs for the parent prefixes against the reuse distance without the MEPs. Since the default root prefix ‘*’ is impossible to cache, we exclude the root prefix from the reuse distance studies. As shown in Figure 3, the reuse distance for the prefix only is much shorter than that with the MEPs. For example, in order to cover 13M IP lookups, MEP needs about 160 entries while only 80 entries are needed if we can cache each prefix with no violation of LPM. Although parent prefix with many children is difficult to cache, this ideal upper bound is still encouraging for the effort to caching the parent prefix. Another interesting observation is that although referencing to the prefixes does show good locality at short reuse distances, the reuse distance is leveling off after about 200. Therefore, the cache size needs to be sufficiently large to capture the reuse of the prefixes.

<table>
<thead>
<tr>
<th>Children</th>
<th>1</th>
<th>2</th>
<th>3-10</th>
<th>&gt;10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parent</td>
<td>4611</td>
<td>3232</td>
<td>4667</td>
<td>2325</td>
</tr>
<tr>
<td>Total MEP</td>
<td>14800</td>
<td>7978</td>
<td>26249</td>
<td>83288</td>
</tr>
<tr>
<td>Average MEP</td>
<td>3.21</td>
<td>2.47</td>
<td>5.62</td>
<td>35.82</td>
</tr>
</tbody>
</table>

Table II. Parent prefixes and MEPs.

### III. Greedy Cache with Prefix Upgrade

The proposed greedy prefix cache improves the basic approach of the prefix cache with MEPs. Upon a cache miss, the search through the trie determines the longest matching prefix. Any matched leaf prefix is always placed in the prefix cache based on the traditional LRU replacement policy. However, when the longest match is a parent prefix, only the associated MEP is cached. The MEP is the shortest expanded disjoint child that matches the destination IP address. It can be formed by taking the bit string up to the level where the longest match to the parent prefix is determined, plus one additional bit from the next bit in the IP address followed by a wildcard ‘*’. For instance, in Figure 2 (b), searching for address <0101...> ends at node <010*> since there is no right link. The MEP can thus be formed as <0101*->Port A>. In other words, the MEP represents the largest sub-tree that matches the IP address without conflicting with any other longest matched prefix within the parent sub-tree.

Instead of the MEP, the parent prefix can be cached when all the children prefixes are also present in the cache. For simplicity, we only consider the parent prefix with a single child prefix. Furthermore, to avoid cache pollution, the parent prefix can only enter the cache to replace its own MEPs. We refer this approach as a prefix upgrade. Consider again the example in Figure 2. Upon a miss to the IP address <0100...>, the search through the trie find the longest match <0100*-Port B> along with a nested parent <01*-Port A>. When inserting <0100*-Port B> into the cache, a search for any existing MEP is carried out. A cached prefix with <01...> is a MEP for <01*-Port A>, and is upgraded (replaced) by the parent prefix. When there is more than one MEP, all the remaining MEPs are also invalidated. In case that no MEP exists in the cache, the parent prefix is not cached.

The prefix upgrade mechanism may not always upgrade an existing MEP to a parent prefix. Figure 4 illustrate another trie example, where the parent prefix <01*->Port A> has two children, <010000000*->Port B> and <0101111*->Port C>. When a request <010000001...> arrives, the node <01000000*> is the last one visited and an MEP <010000001*->Port A> is cached. Next, when <01000000...> comes, it matches <01*->Port A> and then matches <010000000*->Port B>. However, since the parent prefix <01*->Port A> has more than one child, we cannot cache it with the child <010000000*->Port B> along. Nevertheless, we can cache <0100*->Port A> with <010000000*->Port B> since <010000000*->Port B> is the only child from the newly defined parent <0100*->Port A>. Now, because the MEP <010000001*->Port A> is already in the cache, we can upgrade it to the new upgradeable parent <0100*->Port A>.

To create an upgradeable parent is straightforward during the trie traversal. After the matched parent <01*->Part A>, the upgradeable parent is moved to the node after a node with both active left and right links in the traversal path. In Figure 4, for instance, the upgradeable parent is moved from <01*->Part A> to <0100*->Port A> because the node <010*> indicate there exists at least one child prefix in each

![Figure 3. Reuse distances of Prefix-only and with MEP](image-url)
of the left and the right sub-tree. The largest cacheable sub-tree can only start from the first node after <010*>.

The prefix cache placement and replacement policies must be adjusted to accommodate the parent prefix. First, a parent prefix must be placed in the same set as its children so that the longest match requirement can be satisfied without additional searches. Second, the children cannot be replaced before the parent. It is relatively simple in handling the first requirement. As long as the set index comes from the high-order bits of the IP address, the parent and children are naturally located in the same set. This is true for the fact that we consider the prefix upgrade for parents with a single child. Both the parent and the child are located closely to the leaf level. Furthermore, we will show in the performance section that the prefix cache has severe conflict misses, hence requires high set-associativity. The second requirement can be satisfied by marking all the children prefixes when their parent is present in the cache. Instead of replacing a marked child, any parent or MEP prefix closest to the LRU is replaced. One extra bit is added for marking these children.

IV. PERFORMANCE EVALUATION

We use the hit/miss ratios as the major parameter to evaluate the prefix cache performance. The average routing decision time will be shortened if the hit ratio is improved. As a result, the network processor can handle more incoming requests to achieve higher throughput. In addition, upon a cache hit, the search through the trie can be avoided; hence the power can also be saved.

To understand the general performance of the prefix cache, we first simulate the MEP-only caches [1] with various sizes and set-associativities. The results are obtained from simulating the 24-hour trace of MAWI [18] using the as1221 routing table [3]. The trace contains 23.5 million addresses and the routing table has 215451 prefixes. In figure 5, the miss ratios of three cache sizes with 256, 512 and 1024 prefixes are plotted. The set-associativity starts with 16-way and continues all the way to fully-associative. The results clearly indicate both the size and set-associativity are very essential to the cache performance. As illustrated in Figure 3, the reuse distance was leveling off after about 200, it is important to build large enough caches to capture the reuse of the prefixes. Moreover, the results also show that the set-associativity is equally important. The performance continues to improve almost all the way to fully-associative, especially for small caches. In this study, we use the second domain ID (the 8th-15th bits) as the index bits. With the wildcard in prefix, it is reasonable and necessary to index the set using some higher-order bits from the IP address. Although more complicated cache indexing algorithm with index randomization can be applied here, we found that the impact to the overall cache performance is rather minor.

We now present the greedy prefix cache performance and compare it against the MEP cache without caching the parent prefixes. Figure 6 shows the miss ratios of both the MEP and the greedy caches for caches ranging from 128 to 512 prefixes. In this simulation, we consider a fully-associative cache. We can observe that the upgrade mechanism can reduce the miss ratio by about 6-8% for all cache sizes. Due to the fact that more MEPs have a chance to be upgraded, the improvement is slightly better with bigger caches.
With respect to set-associativity, we simulate the three cache sizes, 256, 512 and 1024 with different set-associativities for both the MEP-only and the greedy caches. Again, from the results shown in Figure 7, set-associativity plays an important role in helping the prefix upgrades. An improvement over 7% is possible with high set-associativities. Due to severe conflicts in set-associative prefix caches, many MEPS are replaced prematurely when the set-associativity is low. This conflict not only impacts the hit/miss ratios as illustrated in Figure 5, it also reduces the benefit for upgrading the MEP. Since high set-associativity is necessary to avoid conflicts, the upgrade scheme works better under this circumstance.

V. RELATED WORK

Early works in studying IP Routing table lookup with cache were reported in [5,6,7]. They use the CPU cache hardware for routing table by mapping IP addresses to virtual addresses [5]. The middle bits are picked to translate into physical addresses while the remaining bits are used as cache tags. One to two orders of magnitude improvement can be achieved comparing to pure software-based routing table lookup implementation with a 16KByte L1 cache and 1-MByte L2 cache. Three special cache designs for network processor are studied [6]. The first design, Host Address Cache (HAC), is identical to a conventional CPU cache, which treats the destination host address as a memory address. The second design, Host Address Range Cache (HARC), allows each cache entry corresponds to a contiguous host address range based on the fact that each routing table entry is a prefix and covers a large portion of address space. The third design, Intelligent Host Address Range Cache (IHARC) tries different hash functions to combine the disjoint host address ranges that have the same lookup results. In [7], based on the locality in the IP traffic it studied, the cache space was divided into two zones. The IP address that matches a short prefix is cached in one zone while the IP address that matches a long prefix is cached in another zone. Both zones are organized as fully-associative cache and both LRU and OPT cache replacement polices are studied.

In [23], caches were proposed to reduce both the searching time and updating time. A new technique called controlled prefix expansion is introduced, which expend all the parent prefixes and some leaf prefixes to several disjointed levels. The expansion is optimized by dynamic programming. Although the difficulty of caching parent prefix is solved by expansion, all the expanded prefixes have to be added into the prefix routing table, which dramatically increase its size.

In [15], three mechanisms are proposed to cache the parent prefixes. Complete Prefix Tree Expansion (CPTE) expands a parent prefix along the entire path so that each node in the trie has either two children or no child. The newly added prefix has the same output port as their parent. The original routing table is transformed into a complete prefix tree so that all lookups would end at a leaf prefixes that is cacheable. No Prefix Expansion (NPE) avoids caching the parent prefix. Finally, Partial Prefix Tree Expansions (PPTE) is a trade-off of the previous two mechanisms, which only expand the prefix at the first level.

In [1], the minimum-expansion prefix (MEP) idea is presented instead of expanding the parent prefix statically and put into the routing table. A MEP is created during searching on the trie and saved in cache when necessary. A MEP is a shortest prefix extended the parent prefix which has no conflict with any child prefix and hence cacheable. Since the prefix routing table is unchanged, it saves the memory space and handles the updates more easily than all previous approach. Our work further extends this approach by upgrading the MEPs to their parent prefixes.

Many other trie-based researches focused on how to reduce or pipeline the memory accesses for the trie visit [11,14,19]. Our approach is orthogonal to their approaches and our cache model can be used in any of their work to improve the average processing time and reduce main memory traffic.

Other recent IP-lookup studies are either TCAM-based [2] or hashing-based [8,22,12,13]. We leave the extension of
our cache design of TCAM-based or hashing-based approaches for future study.

VI. CONCLUSION

We proposed a greedy prefix caching mechanism that improves the previous MEP approach [1] by caching the parent prefixes. An existing MEP is upgraded to its parent prefix to expand the cache coverage when the leaf prefix is also cached. The new greedy cache can be applied to existing trie-based algorithms without any modification to the prefix routing table. The experiment results show the proposed prefix caching mechanism achieves up to 8% miss ratio improvement comparing with the existing prefix caches.

ACKNOWLEDGMENT

This work is supported in part by a research donation from Inter Corp. Dr. Wencheng Lu, Dr. Sartaj Sahni and the reviewers provided very valuable comments.

REFERENCES

[22] H. Song et al, IPv6 Lookups using Distributed and Load Balanced Bloom Filters for 100Gbps Core Router Line Cards, In INFOCOM-09, 2009.