Efficient FIB Caching using Minimal Non-overlapping Prefixes*

Yaoqing Liu, Syed Obaid Amin, and Lan Wang
Dept. of Computer Science, University of Memphis
Memphis, TN, USA

yliu6@memphis.edu, soamin@memphis.edu, lanwang@memphis.edu

ABSTRACT

The size of the global Routing Information Base (RIB) has been increasing at an alarming rate. This directly leads to the rapid growth of the global Forwarding Information Base (FIB) size, which raises serious concerns for ISPs as the FIB memory in line cards is much more expensive than regular memory modules and it is very costly to increase this memory capacity frequently for all the routers in an ISP. One potential solution is to install only the most popular FIB entries into the fast memory (*i.e.*, a *FIB cache*), while storing the complete FIB in slow memory. In this paper, we propose an effective FIB caching scheme that achieves a considerably higher hit ratio than previous approaches while preventing the *cache-hiding* problem. Our experimental results show that with only 20K prefixes in the cache (5.36% of the actual FIB size), the hit ratio of our scheme is higher than 99.95%. Our scheme can also handle cache misses, cache replacement and routing updates efficiently.

Categories and Subject Descriptors

C.2 [COMPUTER-COMMUNICATION NETWORKS]: Miscellaneous

General Terms

Algorithms, Design, Performance

Keywords

Routing, FIB caching, Scalability

1. INTRODUCTION

The global routing table size has been growing rapidly, due to a variety of factors such as an increasing number of edge networks, increased use of multihoming, and finer-grained traffic engineering practices [10]. A direct consequence of this problem is the rapid growth of the forwarding table (FIB) size. Although both trends are disturbing, ISPs are more concerned about the FIB size [4], because the FIB memory in line cards costs much more than the memory in route processors as the former needs to support much higher lookup speed at the line rate (*e.g.*, hundreds of millions of packets per second or higher). Moreover, as the size of FIB memory increases, the FIB lookup time may also increase [13] and the line card may become more power-hungry as well [10].

In the long run, modifying the current routing architecture and protocols seems to be the best solution to these problems [7]. However, such proposals may take a long time to deploy due to the high costs associated with them. Meanwhile, ISPs cannot afford to upgrade all their routers frequently. Zhao *et al.* investigated various

possibilities to mitigate the routing scalability issue, and concluded that FIB size reduction would be the most promising solution from an operator's view [17].

One approach to reducing the impact of large FIBs is to use high-speed memory as a *cache* to store the most popular routes [2, 6, 8, 15], while storing the full FIB in lower-cost memory. The feasibility of this approach, which we call *FIB Caching*, depends on how much locality is in the network traffic. In fact, previous studies [5, 6, 12, 15] have shown that a small number of popular prefixes contribute to most of the observed traffic. The data trace from a regional ISP used in our evaluation also supports this observation. As such, a FIB cache needs to store only a small set of popular prefixes, thus saving a router's high-speed memory, increasing lookup speed, and reducing power consumption.

Although caching has been studied extensively in general, *FIB caching* has its unique set of issues. First, network links forward a huge number of packets every second, which means even a 1% miss ratio could lead to millions of lookups per second in *slow* memory. To minimize this problem, an effective FIB caching scheme must achieve an extremely high hit ratio with a modest cache size. Second, the cache miss problem is especially serious when a router starts with an empty cache, so a good scheme needs to quickly and effectively fill the cache even without prior traffic information. Third, Internet forwarding uses longest-prefix match rather than exact match. If not well designed, a FIB caching scheme may cause a *cache-hiding* problem, where a packet's longest-prefix match in the cache differs from that in the full FIB and thus the packet will be forwarded to the wrong next hop (Section 2.1). Finally, prefixes and routes change from time to time, therefore any practical FIB caching scheme needs to handle these changes efficiently without causing the cache-hiding problem.

We propose a FIB caching scheme that selects and generates a minimal number of *non-overlapping* prefixes for the cache. Because the cached prefixes do not cover any longer prefixes in the full FIB, we do not have the cache-hiding problem. Based on this caching model, we develop a FIB caching update algorithm to systematically handle cache misses, cache replacements and routing updates. Our experimental results show that, for a routing table of 372K prefixes, our scheme achieves a hit ratio higher than 99.95% using a cache size of 20K prefixes (5.36% of the full FIB size), and our scheme outperforms alternative proposals in term of hit ratio. In addition, we fill the *initial* cache with the *shortest* non-overlapping prefixes generated from the full FIB, which significantly increases the hit ratio for the initial traffic. Our evaluation results show that the initialized cache has a hit ratio of 85% for the first 100 packets, compared to 65% for an uninitialized cache.

The remainder of the paper is structured as follows. Section 2 gives an overview of our scheme. Section 3 presents our design

*This work was supported by NSF Grant 0721645.

Figure 1: Design Architecture for FIB Caching

in detail. Section 4 evaluates our scheme and compares it with other approaches. Section 5 discusses related work and Section 6 concludes the paper.

2. DESIGN OVERVIEW

Figure 1 illustrates a router architecture with the proposed FIB caching scheme. The control plane contains the RIB, while the Slow FIB and Cache reside in the data plane. The Slow FIB memory contains a copy of the full forwarding table with all prefix entries and next hop information. The Cache contains only the most popular prefixes driven by data traffic. We place the Slow FIB in the data plane (in the line card) rather than the control plane (in the route processor) so that a cache miss/replacement can be quickly handled. The Slow FIB handles four events *A*, *W*, *M* and *O* representing Route Announcement, Route Withdrawal, Cache Miss and Cache Replacement, respectively. The Route Announcement and Route Withdrawal events are generated as a result of RIB updates, which need to be propagated to the FIB. Cache Miss and Cache Replacement are events from the cache. The former happens when an incoming packet does not have a matching prefix in the Cache. The latter occurs when the Cache is full. In the remainder of this paper, full FIB or FIB refers to the Slow FIB, and operations occur in the Slow FIB unless the location is explicitly stated. Before discussing the operations that take place in the Slow FIB and Cache, we explain the cache-hiding problem and outline our solution for handling it in the following section.

2.1 Cache-Hiding Problem

FIB caching is different from traditional caching mechanisms – even if a packet has a matching prefix in the cache, it may not be the correct entry for forwarding the packet if there is a longer matching prefix in the full FIB. Below we use a simple example to illustrate this cache-hiding problem. For ease of illustration, we use 8-bit addresses and binary representations of addresses in our examples.

Suppose a FIB table contains three prefixes as shown in Table 1(a), and the corresponding cache is empty (not shown). Assume a data packet destined to 10011000 arrives at a router. The router then looks for the longest prefix match in the cache, which has no matching entry (the cache is empty). The router then looks up the full FIB in slow memory and loads the matching entry 1001/4 with the next hop 2 to the cache (Table 1(b)). Now, suppose another data packet destined to 10010001 comes. Then, the router will first check the cache to see if there is a prefix matching the destination IP address. It finds the matching prefix 1001/4 in the cache, and thereby sends the packet to next hop 2. This is, however, incorrect because the real matching prefix for IP address 10010001 should be the more specific prefix 100100/6 with the next hop 1. In other words, the cached prefix 1001/4 “hides” the more specific prefix 100100/6 in the full FIB.

Table 1: FIB entries and Cache Entries (The cache is initially empty and it receives one entry upon the first cache miss.)

(a) FIB Entries			(b) Cache Entries		
Label	Prefix	Next Hop	Label	Prefix	Next Hop
A	10/2	4	B	1001/4	2
B	1001/4	2			
C	100100/6	1			

Figure 2: Selection or generation of a leaf prefix

2.2 Our Solution to Cache-Hiding

To illustrate our solution, we use Patricia Tries (*i.e.*, Radix Tree) [11] to store the slow FIB and cached prefixes. Patricia Trie is a space-optimized tree where the child prefix can be longer than the parent prefix by more than one. It is commonly used to store routing tables in a compact manner. Note, however, that our solution can be applied to any tree based structures.

We cache the most specific *non-overlapping* prefixes that do not hide any longer prefixes in the full FIB to avoid the cache hiding problem. In Table 1(a), *C*’s address space is covered by *B*, so they are *not* non-overlapping prefixes (see Figure 2(a)). As such, we cannot simply load the prefix *B* (1001/4) into the cache, because it will cause a problem for the next packet destined to the address 10010000. Instead, we need to generate a leaf prefix *D* (10011/5) to represent the address space under *B* that does not overlap with *C* (Figure 2(a)) and put it into the cache (Figure 2(b)). *D* (10011/5) has the next hop 2, same as its covering prefix *B* (1001/4). The next packet destined to 10010000 causes a cache miss again and correctly finds a matching prefix *C* (100100/6) with the next hop 1 in the slow FIB (Figure 2(c)), which is then loaded into the cache (Figure 2(d)). We call our approach *FIB Caching using Minimal Non-overlapping Prefixes* because we select or generate only the shortest leaf prefixes needed by the data traffic to minimize the number of cached prefixes.

2.3 Slow FIB Operations

Upon receiving an *announcement* or *withdrawal* event from the RIB, the slow FIB updates the corresponding entry and updates the cache if necessary. The specific operations to update the cache are described in Section 3.5 and 3.6.

Upon a *cache miss* event, the FIB returns a leaf prefix that matches the data packet that caused the cache miss to the cache. A new leaf prefix may need to be dynamically generated if the existing leaf prefixes in the FIB do not match the data packet (Section 3.3). Upon receiving a *cache replacement* message, the FIB will either delete or update an entry according to different scenarios, which will be discussed in Section 3.3 and 3.4 in detail.

2.4 Cache Operations

Cache operations include cache initialization, cache-miss traffic

Figure 3: Workflow for Handling Incoming Data Traffic (the dotted line means that during Cache Replacement, the slow FIB needs to be updated but the flow of operation does not continue beyond that point.)

handling, and cache replacement.

Cache Initialization. Handling the initial traffic is a major concern for deploying cache mechanisms [17]. To address this issue effectively, we fill up the initial empty cache with a set of leaf prefixes from the FIB that cover the most IP addresses. More specifically, breadth-first search is employed to find the shortest leaf prefixes from the slow FIB Trie (up to the cache size). This way we achieve a high hit ratio while avoiding the cache-hiding problem. If a router is able to store the cache content in a non-volatile memory before its restart, it can also use that to fill up the initial cache.

Cache Miss Traffic Handling. Packets experiencing cache miss can be stored in a separate queue and be forwarded once the prefixes from slow FIB memory are installed into the cache.

Cache Replacement. We use the LRU (Least Recently Used) replacement algorithm when a new prefix needs to be installed into a full cache. Our decision is based on the study conducted by Kim *et al.* [6], which shows that the LRU algorithm performs almost as good as the optimal cache replacement algorithm.

3. DESIGN DESCRIPTION

3.1 Workflow for Handling Data Traffic

Figure 3 shows how our cache handles an incoming packet. In the ‘Init’ or initialization phase, we load all FIB entries into the slow FIB. Subsequently, we fill up the entire cache with the leaf prefixes that have the shortest length. For any incoming packet, a longest prefix match is performed on the Patricia Trie of the cache. In the case of a prefix match, the packet is forwarded accordingly and the prefix’s status becomes the “latest accessed” to facilitate the cache replacement policy later.

In the case of a *cache miss*, a lookup is performed in the slow FIB and the packet is discarded if the lookup returns no matching prefix. On the other hand, if the longest matching prefix is a leaf node in the Patricia Trie, it is pushed to the cache. Otherwise, i.e., the prefix is an internal node, a more specific prefix is created and pushed to the cache (Section 3.3). The packet is then forwarded to the corresponding next hop. When pushing any prefix to a *full* cache, the cache replacement mechanism removes the least recently used prefix from the cache and installs the new one.

3.2 Data Structure

Each node in the Patricia Trie of the slow FIB is one of four

types, which may change upon an update. These types help us in keeping the cache, slow FIB and the RIB consistent with each other. The four types are as follows (note that this classification does not apply to the cache): (a) *CACHE_ONLY*: a leaf node that is created on demand as a result of the cache miss event; (b) *FIB_ONLY*: a node derived from the original routing table or RIB update, but the prefix is *not* in the cache; (c) *FIB_CACHE*: a leaf node derived from the routing table and the prefix is in the cache; and (d) *GLUE_NODE*: any other auxiliary node except the above three types.

3.3 Handling Cache Misses

In the case of a cache miss, we perform a longest prefix matching in the slow FIB and may encounter the following three cases: (1) if there is no matching node, then drop the packet; (2) if there is a matching leaf node with the type *FIB_ONLY*, then set the type *FIB_CACHE*, and install the prefix with the corresponding next hop into the cache; and (3) if there is a matching internal node with the type *FIB_ONLY*, then we generate a *CACHE_ONLY* node as described below and install it into the cache.

Suppose P_L and P_R are the left and right child of a node P , respectively, and X is the destination IP address. We generate a *CACHE_ONLY* node with the same next hop as its parent on the trie and a prefix containing the first $l + 1$ bits of X , where l is defined as follows: (a) if P_L is NULL, then compare P_R with X to get the common portion Y with length l ; (b) if P_R is NULL, then compare P_L with X to get the common portion Y with length l ; and (c) if P_L and P_R are not NULL, then compare X with P_L and P_R separately, and get the common portion Y_L and Y_R , then find the longer prefix Y with length l from Y_L and Y_R .

Now we provide a detailed example of Case *c* mentioned above. In Figure 4(a), the matching node B with prefix 1001/4 has both a left child (C) and a right child (D). So $Y_L = (X \& C) = (10010100 \& 100100/6) = 10010/5$ and $Y_R = (X \& D) = (10010100 \& 10011/5) = 1001/4$. Therefore, we pick the longer one, $Y=Y_L=10010/5$ and $l=5$. The prefix of the imaginary parent or glue node (E) is 10010/5 as shown in Figure 4(b) and the new leaf node (F) is $X/(l+1)=100101/6$ as shown in Figure 4(c). F ’s node type is *CACHE_ONLY*, as it is generated on demand and will be installed into the cache. Figure 4(d) and 4(e) show the cache entries before and after the update.

3.4 Handling Cache Replacement

When the cache becomes full, some prefixes in the FIB cache need to be evicted according to the LRU replacement strategy. We first remove the prefix from the cache and then update the slow FIB. Two cases may happen:

(1) If the corresponding node type is *CACHE_ONLY*, it means that the node was created on-demand and there would be no such entry in the RIB, so we can remove it directly.

(2) If the corresponding node type is *FIB_CACHE*, it means that this node was originally from the RIB or RIB update, so we cannot remove it completely from the FIB, therefore we change the type to *FIB_ONLY*.

Figure 5 shows a Cache Replacement event on prefix 100100/6 (the *FIB_CACHE* case). Figure 5(a) and 5(b) show the cache operations. Figure 5(c) and 5(d) show the slow FIB operations.

3.5 Handling Route Announcements

In this section and Section 3.6, we discuss the handling of route announcements and withdrawals, respectively (the complete workflow for both is depicted in Figure 6). A route announcement may add a new prefix or update an existing entry in the slow FIB. Below

Figure 4: Example of Cache Miss Update. There are three fields for each node from left to right: prefix, next hop and node type (F: FIB_ONLY, H: FIB_CACHE, C: CACHE_ONLY and G: GLUE_NODE) in the FIB trie. A bold font denotes a field updated in the current step. A solid rectangle denotes a node with a prefix from the original routing table or an update. A dashed rectangle denotes a generated node due to cache miss update. A grey node denotes a node in the cache.

Figure 5: Example of Cache Replacement Update

Figure 6: Workflow for Handling Announcements and Withdrawals (loopbacks to the ‘Listen’ state are not shown.)

we describe each scenario in detail.

When adding a new node to the FIB trie, we need to handle the following two cases.

1. The new node is added as a leaf node: if its direct parent node type is *CACHE_ONLY* (i.e., the prefix of this node was generated on demand and is in the cache), then we remove the parent node from both the FIB and the cache, in order to avoid the cache-hiding problem. If the direct parent of the new node is a *FIB_ONLY*, nothing needs to be done, because the parent node must not be in the cache. If the direct parent of the new node is *FIB_CACHE* (i.e., the prefix attached to the parent node is in the cache, and needs to be removed from there), then we set the parent node type as *FIB_ONLY* and remove the prefix from the cache.
2. The new node is added as an internal node: all the *CACHE_ONLY* nodes whose next hops are derived from this node should have the same next hop as this one, so we update these nodes with the new next hop and update the

cache accordingly to synchronize the next hop information. We do not update the *FIB_CACHE* nodes because their next hops are directly from the corresponding real prefixes in the RIB, not derived from their ancestors.

Similarly, we need to handle two cases when updating an existing FIB entry to change its next hop value (see below).

1. The existing node is a leaf node: if the node type is *FIB_ONLY* (i.e., the prefix is not in the cache), we simply update it with the new next hop. Otherwise, we update it with the new next hop, set its type as *FIB_CACHE*, and update its next hop in the cache accordingly.
2. The existing node is an internal node: we update all the *CACHE_ONLY* nodes whose next hops are derived from this node with the new next hop, and update the cache accordingly.

Figure 7 depicts an Announcement example, in which an update with the next hop 3 comes for an existing node (*E*) with the prefix 10010/5. As the update is for an internal node, the node type will be changed to *FIB_ONLY*. Moreover, we update those *CACHE_ONLY* nodes whose next hop was derived from this node, in order to keep the forwarding behavior correct. For example, in Figure 7(a), node *F* with the prefix 100101/6 is a *CACHE_ONLY* node that inherited its next hop from *E*, so its next hop is changed to 3 in Figure 7(b), and we update the corresponding entry in the cache as shown in Figure 7(c) and Figure 7(d). This way, any subsequent data packet destined to 100101/6 will be forwarded correctly to the new next hop 3.

3.6 Handling Route Withdrawals

For a route withdrawal, we need to process it only if it matches an existing node in the FIB, since the corresponding prefix is supposed to be in the FIB in order to be “withdrawn”. The matching node can be either a leaf node or an internal, and we process it as follows. (1)

Figure 7: Example of Announcement Update

Figure 8: Example of Withdrawal Update

Leaf node: If the node type is *FIB_CACHE*, we delete it from both the FIB and the cache. If the node type is *FIB_ONLY*, we delete it from the FIB only, since it is not in the cache.

(2) Internal node: we delete its next hop and change the node type to *GLUE_NODE* (it is still useful in the FIB trie to connect the other nodes). Since our algorithm puts only leaf nodes in the cache, this internal node cannot be in the cache and therefore no cache update is needed. Then, we update the *next hop* field of those *CACHE_ONLY* nodes whose next hop was derived from this node. Finally, we update the cache accordingly.

Figure 8 shows an example of removing the prefix 1001/4 (the internal node case). First, the type of node *B* (1001/4) is set to *GLUE_NODE*. The next step is to update the affected *CACHE_ONLY* nodes. In this example, node *D* (10011/5) is the only affected *CACHE_ONLY* node. It is updated with the next hop (4) of its nearest ancestor *A* as shown in Figure 8(b). Subsequently, the cache entry *D'* with prefix 10011/5 is updated with the new next hop 4 as shown in Figure 8(d).

4. EVALUATION

To evaluate our scheme, we used a 24-hour traffic trace of more than 4.1 billion packets from a regional ISP collected from 12/16/2011 to 12/17/2011. We obtained routing tables and updates of 30 different routers from the route-views2 data archive [1] on 12/16/2011 and 12/17/2011. After the initialization of the slow FIB and cache, we run our caching scheme with the data and updates. The updates and data are also passed through an emulated router without the cache to verify the forwarding correctness of our scheme. Our results are similar for all the 30 routers, so we present the results from one of them in most cases.

4.1 Traffic Distribution

Figure 9 shows the traffic distribution over the prefixes from the forwarding table of one of the thirty routers. The x-axis represents the popular prefix rank, and the y-axis represents the cumulative percentage of the IP packets covered by the popular prefixes. We make two main observations: (a) the top 10, 100, 1K, 10K, 20K popular prefixes cover about 42.79%, 79.18%, 93.81%, 99.51%,

Figure 9: Traffic Distribution On Non-overlapping Prefixes

and 99.87%, respectively, of the traffic, which supports a common finding from several other studies [5, 6, 12, 15], i.e., a very small number of popular prefixes contributes to most of the traffic; and (b) most of the entries in the global routing tables are not in use during this period. In fact, more than 70.18% of the FIB entries were not used at all, which further suggests the feasibility to introduce an efficient caching mechanism for the routers.

4.2 Hit Ratio

The *hit ratio* of a cache is the success rate of finding a matching entry in the cache. It is considered one of the most important metrics to evaluate a caching scheme. For a given cache size, the higher the hit ratio is, the better the cache scheme would be. In our experiments, we obtain the hit ratios for 30 routers with different cache sizes ranging from 1K to 20K prefixes. Figure 10(a) shows different hit ratios for one router with five different cache sizes over the 24-hour period. We observe that on average the hit ratio is 96.83%, 98.52%, 99.25%, 99.84%, 99.95% for the cache size of 1K, 2K, 3.5K, 10K and 20K, respectively. The dips around 870 million data packets is due to the traffic pattern around 7:30am, which has the lowest traffic rate but a similar number of distinct destination addresses. This leads to a high miss ratio, as we are dividing roughly the same number of cache misses with a much lower number of packets. Furthermore, we found that the hit ratio

Figure 10: Different Hit Ratios

tends to be more stable when cache size increases. Other routers have very similar results as this one.

We also compared our scheme with different cache-hiding approaches. The most straightforward one is the *Atomic Block* approach, which loads not only a matching prefix into the cache, but also finds all the sub-prefixes of the matching prefix in the FIB and loads them into the cache, so that subsequent packets will not encounter the cache-hiding problem. Another method *Uni-class* divides up a matching prefix into multiple fixed-length (24 bits) sub-prefixes on the fly and installs the one matching the destination address into the cache [6]. This approach assumes that 24 is the longest prefix length in the FIB so the cached $/24$ prefixes will not hide more specific prefixes in the FIB. This assumption is usually true as operators filter out prefixes longer than $/24$ to prevent route leaks. Figure 10(b) compares the hit ratios for the different caching approaches with a fixed cache size of 20K. Our approach has a 99.95% hit ratio on average. The Atomic Block approach has a 99.62% hit ratio and Uni-class approach has a 97.19% hit ratio, on average. Although the hit ratio of the Atomic Block approach is close to our approach, it takes much more time to maintain the cache as shown in Section 4.5. The difference between the hit ratios of the Atomic Block approach and our scheme is due to the fact that the Atomic Block approach fills the cache with all the sub-prefixes of a matching prefixes, which may include many prefixes that will not match subsequent packets. On the other hand, our scheme creates only the most specific prefix that matches an arriving packet's destination address and thus, for a given cache size, our scheme covers more useful prefixes than the Atomic Block approach. The low hit ratio of the Uni-class approach is due to its fixed long prefix length (24). Given the same cache size, it can cover much fewer useful addresses than the other approaches.

Moreover, we compared our approach with three techniques proposed by Liu [8], CPTE, NPE and PPTE, using a static routing table (the author did not specify update handling algorithms). NPE does not increase the FIB size and has a 99.16% hit ratio on average. PPTE increases the FIB size by 13,384 and has a 99.48% hit ratio on average. CPTE expands the FIB trie into a complete binary tree and installs disjoint prefixes into the cache, thus it has the same hit ratio as our scheme (not shown in the figure), but it significantly increases the FIB size by more than two times from 371,734 to 1,131,635 prefixes. In our scheme, we only increase the full FIB size by 6,288 and reach a hit ratio of 99.95% on average. Finally, the RRC-ME algorithm proposed by Akhbarizadeh et al. [2] uses a binary tree (with no expansion) and only installs or generates a disjoint prefix into the cache on the fly, and it will have the same hit ratio as our scheme (not shown in the figure), but our update handling algorithm is much more efficient (Section 4.4).

Figure 11: Update Handling Impact

4.3 Initial Traffic Handling

One of the biggest concerns for ISPs is how to handle the initial traffic when the cache starts with an empty set [17]. Instead of a cold start, we fill the initial empty cache completely with the shortest non-overlapping prefixes if there is no history of popular prefixes available. Figure 10(c) shows the initial traffic hit ratios. We used the first 1 million packets to do the experiment. The top line represents the hit ratio with cache initialization and the lower line represents the one without cache initialization. After the first 100 packets, the initialized cache has a hit ratio of 85% and the uninitialized one has a hit ratio of only 65%. Their hit ratios are very close to each other once 100,000 packets are forwarded.

4.4 Routing Update Handling Performance

Figure 11 shows the routing update handling performance. The top curve represents the total number of RIB updates. The middle curve represents the total number of updates (8,348) pushed to cache including next hop changes (8,251) and prefix deletions, while the bottom curve shows the number of prefix deletions (97), which is only 3.18% of total number of RIB updates. Since very few updates are pushed to the cache, the updates almost have no influence on the cache hit ratio. On the other hand, in the RRC-ME approach [2], each updated prefix needs to be converted into two IP addresses first and they are looked up in the cache to find out the matching prefix. In the process, the cache will be interrupted twice if there is no matching prefix; otherwise, it gets three interruptions. Specifically, in the period of 24 hours, the previous work needs at least 523,754 ($261,877 \times 2$) lookups of the cache as compared to our scheme that needs only 8,251 lookups.

4.5 Time Cost

Figure 12 compares the time cost to process all the routing updates and data of three approaches. We made two main observa-

Figure 12: Total Time Cost

tions: (a) the Atomic Block approach takes about four times longer to finish the same task than the other two approaches; (b) our approach takes almost the same time as the Uni-class approach, but our approach has a much higher hit ratio as shown previously.

5. RELATED WORK

Liu proposed Routing Prefix Caching for network processors [8], which employs three prefix expansion methods, NPE, PPTE and CPTE. These solutions can eliminate the inter-dependencies between prefixes in the cache, but they either will increase the FIB size considerably or have a high miss ratio. Akhbarizadeh *et al.* proposed RRC-ME [2]. This solution can also solve the cache-hiding problem through using disjoint prefixes, but it has significant update handling overhead, especially in the worst cases. Kim *et al.* proposed route caching using flat and uniform prefixes of 24 bits long [6]. It can reach fast access speed using a flat hash data structure for lookup. However, this approach leads to prefix fragmentation and thus has a lower hit ratio than our approach as shown in our evaluation results.

FIB aggregation can reduce the FIB size by aggregating a large FIB table into a smaller one with the same forwarding results. There have been a number of FIB aggregation algorithms [3, 16, 9, 14]. Their results show that the FIB size can be reduced to at most 30% of the original table size. FIB caching is complementary to FIB aggregation. In fact, the full FIB can be aggregated first and then serve as the basis for caching, which can further reduce the required cache size.

The Virtual Aggregation (VA) scheme [3] tries to install some virtual prefixes which are shorter than real prefixes, such as /6, /7 and /8, to legacy routers to control FIB size growth. It can reduce the FIB size on most routers, while the routers that announce the virtual prefixes still need to maintain many more specific prefixes. Our FIB caching scheme can be applied to those routers with a larger FIB size in a network deploying VA.

6. CONCLUSION

We presented an effective caching scheme to mitigate the problems caused by the rapidly increasing size of the global forwarding table. This scheme allows ISPs to keep their operational cost low by storing a fraction of the full FIB in the expensive fast memory, while storing the full FIB in slow memory. Our results based on real data show that we can use only 3.5K prefixes to reach a hit ratio of 99.25% and 20K prefixes to reach a hit ratio of 99.95%. Moreover, we fill the initial empty cache with the shortest non-overlapping prefixes and obtain a high hit ratio for the initial traffic. Finally, our scheme includes a set of efficient algorithms to process both FIB and cache update events, while preventing the cache-hiding problem.

7. ACKNOWLEDGMENT

We thank Christos Papadopoulos, Daniel Massey and Kaustubh Gadkari from Colorado State University for sharing their traffic trace, as well as the anonymous reviewers for their feedback.

8. REFERENCES

- [1] Advanced Network Technology Center at University of Oregon. The RouteViews project. <http://www.routeviews.org/>.
- [2] M. J. Akhbarizadeh and M. Nourani. Efficient prefix cache for network processors. In *Proceedings of the High Performance Interconnects*, 2004.
- [3] R. Draves, C. King, S. Venkatachary, and B. D. Zill. Constructing Optimal IP Routing Tables. In *Proc. IEEE INFOCOM*, 1999.
- [4] V. Fuller. Scaling Issues with Routing+Multihoming. <http://www.vaf.net/~vaf/apricot-plenary.pdf>, 1996.
- [5] K. Gadkari, D. Massey, and C. Papadopoulos. Dynamics of prefix usage at an edge router. In *Proceedings of the 12th international conference on Passive and active measurement*, PAM'11, 2011.
- [6] C. Kim, M. Caesar, A. Gerber, and J. Rexford. Revisiting route caching: The world should be flat. In *Proceedings of the 10th International Conference on Passive and Active Network Measurement*, PAM '09, 2009.
- [7] T. Li. Preliminary Recommendation for a Routing Architecture. RFC 6115, 2011.
- [8] H. Liu. Routing prefix caching in network processor design. In *Tenth International Conference on Computer Communications and Networks*, 2001.
- [9] Y. Liu, X. Zhao, K. Nam, L. Wang, and B. Zhang. Incremental forwarding table aggregation. In *Proc. IEEE Globecom*, 2010.
- [10] D. Meyer, L. Zhang, and K. Fall. Report from the IAB Workshop on Routing and Addressing. RFC 4984, 2007.
- [11] D. R. Morrison. PATRICIA-practical algorithm to retrieve information coded in alphanumeric. *J. ACM*, 15(4):514–534, Oct. 1968.
- [12] J. Rexford, J. Wang, Z. Xiao, and Y. Zhang. BGP routing stability of popular destinations. In *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurement*, IMW '02, 2002.
- [13] G. Trotter. Terminology for Forwarding Information Base (FIB) based Router Performance. RFC 3222, 2001.
- [14] Z. A. Uzmi, M. Nebel, A. Tariq, S. Jawad, R. Chen, A. Shaikh, J. Wang, and P. Francis. SMALTA: practical and near-optimal FIB aggregation. In *Proceedings of the Seventh Conference on emerging Networking EXperiments and Technologies*, CoNEXT '11, 2011.
- [15] W. Zhang, J. Bi, J. Wu, and B. Zhang. Catching popular prefixes at as border routers with a prediction based method. *Comput. Netw.*, 56(4):1486–1502, Mar. 2012.
- [16] X. Zhao, Y. Liu, L. Wang, and B. Zhang. On the Aggregatability of Router Forwarding Tables. In *Proc. IEEE INFOCOM*, 2010.
- [17] X. Zhao, D. J. Pacella, and J. Schiller. Routing scalability: an operator's view. *IEEE Journal on Selected Areas in Communications*, 28(8):1262–1270, Oct. 2010.