Towards TCAM-based Scalable Virtual Routers

Layong Luo[*†], Gaogang Xie[*], Steve Uhlig[‡],
Laurent Mathy[§], Kavé Salamatian[¶], and Yingke Xie[*]

[*]Institute of Computing Technology, Chinese Academy of Sciences (CAS), China
[†]University of CAS, China, [‡]Queen Mary, University of London, UK
[§]University of Liège, Belgium, [¶]University of Savoie, France
{luolayong, xie, ykxie}@ict.ac.cn, steve@eecs.qmul.ac.uk,
laurent.mathy@ulg.ac.be, kave.salamatian@univ-savoie.fr

ABSTRACT

As the key building block for enabling network virtualization, virtual routers have attracted much attention recently. In a virtual router platform, multiple virtual router instances coexist, each with its own FIB (Forwarding Information Base). The small amount of high-speed memory in a physical router platform severely limits the number of FIBs supported, which leads to a scalability challenge. In this paper, we present a method towards TCAM (Ternary Content Addressable Memory) based scalable virtual routers, through a merged data structure that enables the sharing of prefixes from several FIBs in TCAMs. Based on this data structure, we propose two approaches to merge multiple FIBs in TCAMs, paving the way for scalable virtual routers. Experimental results show that, by using the two approaches for storing 14 full IPv4 FIBs, the TCAM memory requirement can be reduced by about 92% and 82% respectively, compared with the conventional approach of treating FIBs as independent entities.

Categories and Subject Descriptors

C.2.6 [**Internetworking**]: Routers

General Terms

Algorithms, Design, Experimentation, Performance

Keywords

Virtual routers, TCAM, FIB completion, FIB splitting

1. INTRODUCTION

Network virtualization [5] is a promising way to achieve cost-efficient utilization of networking resources, to support customized routing, *e.g.*, for application-specific forwarding, customer-based routing, and to enable experimentation and deployment of new network protocols, *e.g.*, CCN, without interrupting normal network operations. Indeed, network virtualization allows multiple virtual networks to coexist on an underlying shared substrate, in isolation from each other.

As the key building block for enabling network virtualization, virtual routers have attracted much attention in recent years [4, 6–10, 18, 19, 22]. When multiple virtual router instances are supported on the same physical equipment, we will use the term *virtual router platform*. In a virtual router platform, multiple virtual router instances coexist and each has its own FIB (Forwarding Information Base). To achieve high forwarding performance, these FIBs are preferably stored in high-speed memory, such as TCAMs (Ternary Content Addressable Memory) or SRAMs (Static Random Access Memory). However, due to physical limitations (*e.g.*, cost, power, and board space), high-speed memory turns out to be a scarce resource, and its size is limited.

On the other hand, with the growing demand for virtual routers, the number of virtual router instances supported in a virtual router platform is expected to keep increasing, with the size of each FIB also expected to grow. This creates a high requirement for large high-speed memory. The gap between the high demand and the limited memory size will create a scalability issue on the virtual router platform. That is, the small amount of high-speed memory severely limits the number of virtual router instances supported. For example, a very large TCAM on today's market has up to 1M entries [2], while a full IPv4 FIB contains about 400K prefixes currently [1], which means that only two FIBs can be stored in such a device.

Efforts in two areas can be made to improve the scalability of a virtual router platform. First, the size of the high-speed memory can be increased to support as many virtual router instances as possible. However, this effort usually results in high power consumption and high cost, which makes it unsustainable in the long term. Second, the amount of memory required for storing multiple FIBs can be reduced by merging and compressing FIBs, so that more FIBs can fit in a given memory size, without the needs to increase system power and cost. We believe this latter approach to be more viable.

In the last few years, the scalability challenge brought by virtual routers has created interest from the research community, and previous work [7–10, 18] mainly focused on SRAM-based scalable virtual routers. Tree-based (*e.g.*, trie or 2-3 tree) algorithms have been proposed to merge multiple FIBs into a single tree, and thereby many FIBs can be efficiently stored in the SRAMs. However, none of the previous work was targeted towards using TCAMs to build scalable virtual routers.

TCAMs are popular and promising devices to build IP lookup engines, thanks to their deterministic high performance and simplicity [12, 25]. TCAMs always guarantee a deterministic high lookup throughput of one lookup per clock cycle. Although an SRAM may run at a higher clock frequency than a TCAM (thus requiring less time for each memory access), SRAM-based solutions usually have variable lookup performance, and require multiple memory accesses for one lookup in the worst case [16]. TCAMs have traditionally exhibited higher power consumption and lower densities than SRAMs, although as semiconductor technology improves and demand increases, high-capacity, very high-speed TCAMs have become available. For example, the NL9000 family TCAMs [2] from NetLogic Microsystems, which contain up to 1 million 40-bit entries, can perform up to 1200 million decisions per second. All this suggests that TCAMs should also be a viable option to build virtual routers.

In this paper, we propose a merged data structure to share prefixes from individual FIBs in TCAMs. We propose two approaches based on this data structure to build TCAM-based scalable virtual routers. The first approach called FIB completion, merges the prefixes from all the FIBs into one TCAM. This approach exhibits the best scalability, but a large worst-case update overhead. The second approach, called FIB splitting, mixes the advantages of merged and non-shared data structures by splitting the prefixes into two prefix sets. This approach also yields very good scalability, with a much more reasonable worst-case update overhead. We provide experimental results showing that, for 14 full IPv4 FIBs, the proposed approaches can achieve a TCAM memory reduction of 92% and 82%, respectively, compared with a non-shared approach. This suggests that VPNs (Virtual Private Network) and experimental network testbeds can also benefit from our approaches.

The rest of the paper is organized as follows. In Section 2, we discuss the background on TCAM-based lookup engines and describe the non-shared approach for TCAM-based virtual routers. In Section 3, we describe the merged data structure that is the basis of this work. In Sections 4 and 5, we present two approaches based on the merged data structure. We evaluate the two approaches and compare them with the non-shared approach in Section 6. We discuss the related work in Section 7 and conclude in Section 8.

2. BACKGROUND

A TCAM is a fully associative memory, in which each bit of an entry can be specified in one of three states: ‘0’, ‘1’ and ‘X’ meaning “do not care”, also called wildcard bits. Thanks to the wildcard bits, the TCAM is very suited for storing IP prefixes in IP lookup engines. Given a destination IP address, all the prefixes stored in the TCAM can be compared with the IP address simultaneously. Indeed, multiple prefixes may match the IP address, and the matched prefix at the lowest address will be selected as the only matching result, as the TCAM entries have an intrinsic priority, *i.e.*, an entry at a lower address has a higher priority. Although some TCAMs may have different priority logics, we assume in this paper that the TCAMs we use hereafter implement the above priority. After the matching result is determined, the TCAM returns the address of the matched prefix. The TCAM usually completes an IP lookup every clock cycle, resulting in consistent very high lookup speed.

FIB 0

prefix	next hop
0*	A1
1*	A2
00*	A3
100*	A4
101*	A5
111*	A6

(a)

FIB 1

prefix	next hop
0*	B1
1*	B2
11*	B3
100*	B4
101*	B5
111*	B6

(b)

Figure 1: Two sample FIBs

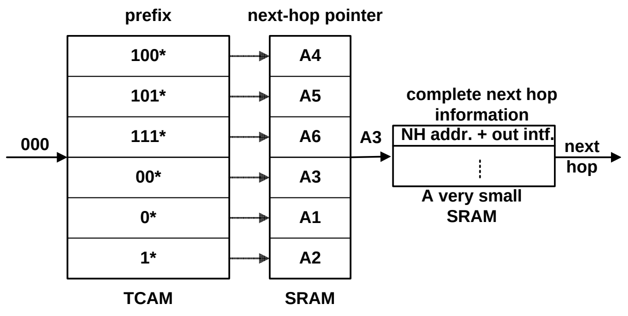


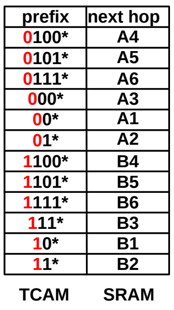
Figure 2: A TCAM-based IP lookup engine

2.1 Traditional TCAM-based IP Lookup

To fix ideas, let us start by considering a traditional router, which is a platform hosting a single router instance, with only one FIB. Each entry in the FIB is comprised of an IP prefix and its corresponding next hop (NH). Figure 1(a) shows a sample FIB.

In a typical TCAM-based IP lookup engine, the IP prefixes are stored in the TCAM in decreasing prefix-length order for correct longest prefix matching (LPM) [17]. The NH pointers are stored in an associated SRAM, and the complete NH information is stored in another very small SRAM. Figure 2 shows a TCAM-based IP lookup engine containing the FIB shown in Figure 1(a). Generally, a complete NH consists of the IP address of the NH and the output interface, consuming about 5 bytes per NH. The number of unique NHs is small due to the limited number of interfaces in a router. One can assume that this number is less than 256 [7], so that a 1-byte NH pointer can represent the NH. Therefore, storing the NH pointers in the associated SRAM instead of the complete NHs consumes much less memory. The complete NHs can be stored in another very small SRAM, and the NH pointers are used to locate the final complete NHs (see Figure 2). The total size of the complete NHs is the same for all approaches, and is so small that we will ignore it in the remainder of the paper. Hereafter, for simplicity, we will use the term “NH” to mean “NH pointer”, and focus our attention only on the TCAM and its associated SRAM storing NH pointers in the TCAM-based IP lookup engine.

In the TCAM-based IP lookup engine shown in Figure 2, given destination IP address 000 to lookup, the two prefixes 00* and 0* are matches, and the address of prefix 00*



prefix	next hop
0100*	A4
0101*	A5
0111*	A6
000*	A3
00*	A1
01*	A2
1100*	B4
1101*	B5
1111*	B6
111*	B3
10*	B1
11*	B2
TCAM	SRAM

Figure 3: A non-shared approach for virtual routers

will be returned by the TCAM, because it is stored at a lower TCAM address (*i.e.*, a higher priority). Then, this address is used to find the corresponding NH pointer A3 in the associated SRAM. Finally, the complete NH is obtained in another SRAM through pointer A3, and the packet is forwarded towards this NH.

2.2 Naive Approach for Virtual Routers

In the context of virtual routers, multiple FIBs are hosted in a physical platform. One way to arrange prefix sets from several FIBs in one TCAM is to put each prefix set in a separate region. We will call this approach the “non-shared approach”, as identical prefixes from different FIBs are not shared. We use the two FIBs shown in Figure 1 to illustrate this non-shared approach for TCAM-based virtual routers.

In a virtual router platform, a Virtual router ID (VID) identifies each router and its FIB. Assume that VID 0 is assigned to FIB 0 and VID 1 is assigned to FIB 1 in Figure 1. Before storing the two FIBs in the TCAM, the VID is prepended to each prefix to form a virtual prefix. For example, the virtual prefix for prefix 0* in FIB 0 is 00*, while the corresponding virtual prefix is 10* for the same prefix in FIB 1. After virtual prefixes are formed, the virtual prefix sets from different FIBs can be directly stored in a TCAM without interfering with each other, as the VID for each FIB is unique. Then, their corresponding NHs are stored in an associated SRAM. Figure 3 illustrates one way of storing FIB 0 and FIB 1 using this approach.

In the non-shared approach, the IP lookup process is as follows. For an incoming packet, the destination IP is extracted from the packet header and the VID is determined from contextual information (*e.g.*, packet header, virtual interface). A Virtual IP address (VIP) is formed by prepending the VID to the destination IP address. Then, the VIP is sent to the TCAM for lookup. For example, given destination IP 000 and VID 1, VIP 1000 is looked up in the TCAM shown in Figure 3, and the virtual prefix 10* is matched. Then, the corresponding NH B1 is found in the associated SRAM. Note that the lookup process in the non-shared approach is similar to that in traditional routers, except for the use of the VIP for TCAM search.

The main issue with the non-shared approach is the TCAM memory requirement, which increases significantly as the number of FIBs increases. For example, in Figure 1, there are 6 entries in each of the two FIBs, and the total number of TCAM entries in the non-shared approach (see Figure 3) is 12, which is the sum of the number of entries in the individual FIBs.

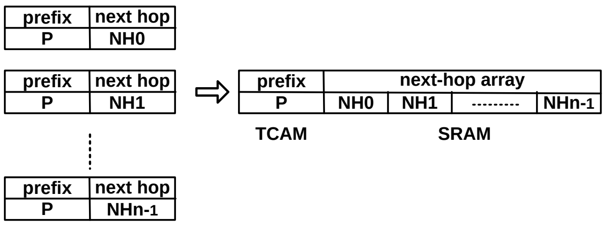
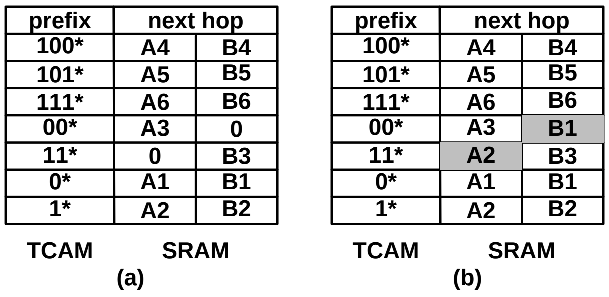


Figure 4: The merged data structure



(a)

prefix	next hop	
100*	A4	B4
101*	A5	B5
111*	A6	B6
00*	A3	0
11*	0	B3
0*	A1	B1
1*	A2	B2
TCAM	SRAM	

(b)

prefix	next hop	
100*	A4	B4
101*	A5	B5
111*	A6	B6
00*	A3	B1
11*	A2	B3
0*	A1	B1
1*	A2	B2
TCAM	SRAM	

Figure 5: (a) The basic merged FIB, and (b) its completed version

3. MERGED DATA STRUCTURE

In the non-shared approach, the total memory space required in the TCAM increases significantly as the number of FIBs increases, and thus the number of virtual router instances supported cannot scale well.

We argue that the prefix similarity among different FIBs can be exploited to significantly reduce the TCAM memory requirement. A key observation is that, if two entries from different FIBs share the same prefix, they can be merged into a single entry in the TCAM-based lookup engines. For example, an entry <P, NH0> in FIB 0 and an entry <P, NH1> in FIB 1 can be merged into a single entry <P, [NH0, NH1]>. The prefix P is stored in the TCAM, and its corresponding NH array [NH0, NH1] is stored in the associated SRAM.

Figure 4 shows the merged data structure for n FIBs in TCAM-based lookup engines. If there is a common prefix P in n FIBs, these n entries can be merged into a single entry, with prefix P associated with a NH array containing n NHs. In the NH array, the i^{th} next hop (NHi) is the one associated with prefix P in the i^{th} FIB.

Based on the merged data structure, the two FIBs shown in Figure 1 can be represented by the merged FIB in Figure 5(a). The prefix set in the merged FIB consists of the union of the prefixes in the individual FIBs, and is henceforth called the Union Prefix Set (UPS). To get the UPS from multiple FIBs, we adopt the trie merging approach proposed in [7].

Initially, an auxiliary 1-bit trie is built from each individual FIB. Figure 6 shows two 1-bit tries, which are built from the two sample FIBs in Figure 1, respectively. In these tries (we will use the terms “1-bit trie” and “trie” interchangeably hereafter), a prefix which does not correspond to any NH in the FIB is associated with an invalid NH (represented as 0). After all the tries are built, they are merged into a merged trie using the strawman approach in [7]. The nodes corresponding to a same prefix in different tries can be merged,

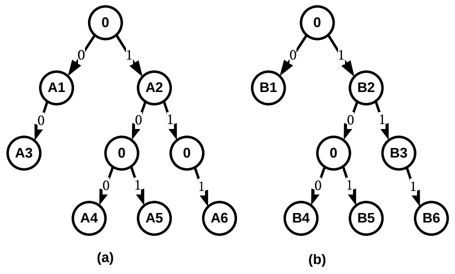


Figure 6: Auxiliary tries for the two FIBs in Figure 1

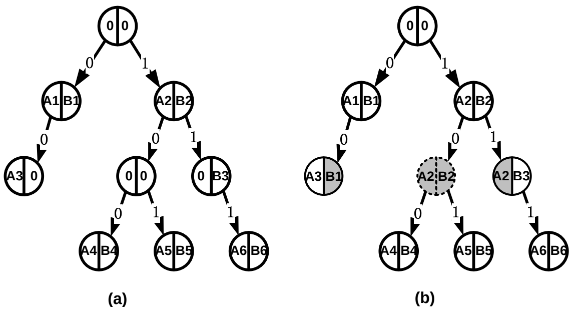


Figure 7: (a) The merged trie, and (b) its completed version

and the NHs for all virtual routers are stored in the merged node. Using this approach, the two tries shown in Figure 6 can be merged into the merged trie depicted in Figure 7(a).

The prefix set represented by the merged trie is the UPS. Note, however, that prefixes represented by merged trie nodes that contain only invalid NHs are not in the UPS, as they do not appear in any of the merged FIBs (*e.g.*, this is the case for prefix 10* in the merged trie in Figure 7(a)).

To clearly show the difference between the merged FIB and the non-shared FIB, we take prefix 100* as an example. We have an entry <100*, A4> in FIB 0 and an entry <100*, B4> in FIB 1. In the non-shared approach (see Figure 3), two separate entries are needed. In the merged FIB, we have a single entry <100*, [A4, B4]>, which denotes that prefix 100* is a prefix in both FIBs, with a corresponding NH A4 for FIB 0, and B4 for FIB 1.

However, a prefix in the UPS does not always exist in all FIBs. For example, prefix 00* exists in FIB 0 but not in FIB 1. In this case, we introduce an invalid NH 0 (null NH). A NH is null when the prefix does not exist in the corresponding individual FIB. For example, prefix 00* does not exist in FIB 1, and therefore the second NH associated with prefix 00* in the merged FIB is denoted as 0 in Figure 5(a).

Using the merged data structure, the number of entries in the merged FIB is significantly lower than the sum of the number of entries in the individual FIBs, which dramatically reduces the TCAM memory requirement. In our example, the number of TCAM entries needed to store the prefixes in the merged FIB (*i.e.*, the UPS) in Figure 5(a) is 7, down from 12 in the non-shared approach shown in Figure 3.

The IP lookup process based on this merged data structure works as follows. The destination IP address is used as the key to be searched in the TCAM. After the search terminates, the address of the matched prefix is used to locate the NH array containing the corresponding n NHs in the associated SRAM. Then, the VID is used as an offset to find the corresponding NH in the NH array. For example, in Figure 5(a), given IP address 100 and VID 1, the first entry in the TCAM will match the IP address, and [A4, B4] is the corresponding NH array. VID 1 is then used as the offset to yield NH B4.

In essence, the proposed method looks up an IP address in all the FIBs simultaneously, and the VID is used to select the relevant NH. While this approach is, at first glance, straightforward, it does introduce inconsistencies and results in incorrect lookups in some cases. For example, given IP address 000 and VID 1, the correct NH should be B1 (see Figure 1(b)). However, if this lookup is performed in the merged FIB shown in Figure 5(a), the NH is 0, which indicates an invalid NH. The reason for this is that the TCAM always returns the first matching result (*i.e.*, the address of the longest matching prefix), while the matched prefix 00* does not exist in FIB 1.

The fundamental issue is that the merged data structure yields a merged FIB that contains all the prefixes appearing in at least one of the individual FIBs (*i.e.*, prefixes from the UPS). This can result in the "artificial" insertion of prefixes in individual FIBs (*e.g.*, insertion of prefix 00* with NH 0 in FIB 1 in Figure 5(a)), while those "added" prefixes can in turn "mask" correct, but shorter-length entries during the matching process (*e.g.*, given IP address 000 and VID 1, the "added" prefix 00* masks the valid prefix 0* in FIB 1 in Figure 5(a)). Note that this issue is specific to FIB merging in a TCAM environment, because of the intrinsic priority matching service provided by such components.

To ensure correct prefix matching, TCAM-based virtual router FIB management methods must *avoid incorrect matching, resulting from the masking of a shorter prefix in an individual FIB by a longer prefix in the merged FIB.*

In the following sections, we present two TCAM FIB merging approaches that avoid the prefix masking issue.

4. FIB COMPLETION

4.1 Completion Approach

Our first TCAM FIB merging approach, called *FIB completion*, addresses the prefix masking issue described in the previous section in a direct way: whenever a prefix from the UPS does not appear in a given individual FIB, we simply associate it with a valid NH in this FIB. The valid NH is the one associated with the longest ancestor prefix in the same FIB. In other words, each null NH entry described in Section 3 is replaced by the first valid NH entry encountered when going up the corresponding trie branch towards the root.

Figure 7(b) shows the merged trie in FIB completion version, which corresponds to the original merged trie shown in Figure 7(a). For example, in Figure 7(a), prefix 11* from the UPS does not appear in FIB 0. In this case, we associate prefix 11* with NH A2, copied from the NH associated with its longest ancestor prefix in the same FIB (*i.e.*, prefix 1* in FIB 0). Therefore, entry <11*, [0, B3]> is replaced by <11*, [A2, B3]>, as A2 is the correct NH according to the LPM rule. Note that this copied NH A2 is drawn in gray in Figure 7(b), which denotes that it is a completed NH, as opposed to a NH originally associated with the corresponding prefix (which we call original NH). Likewise, <00*, [A3, B1]> results from the completion process, so that the longer prefix 00*, not present in FIB 1 and masking the valid prefix 0*, yields the expected NH B1. Note that as prefix 10* does not appear in any of the original individual FIBs (*i.e.*, prefix 10* does not exist in the UPS), this prefix will not be included in the TCAM. We dash the nodes corresponding to prefixes not in the UPS in Figure 7(b). For those prefixes not in the UPS, completion is also performed (*e.g.*, see <10*, [A2, B2]> in Figure 7(b)) to simplify the update process.

Using the FIB completion approach, the merged FIB shown in Figure 5(a) is transformed into the one shown in Figure 5(b). The TCAM-based lookup engine with FIB completion is the same as that described in Section 3. When compared with the merged FIB shown in Figure 5(a), the only difference is that the null NHs in the associated SRAM are replaced by entries yielding correct lookup results.

4.2 Lookup and Update Process

The IP lookup process in the FIB completion approach is exactly the same as the one described in Section 3. However, the update process is different.

For merged trie maintenance purposes, the FIB completion algorithm must keep track of which NH entry exists in the corresponding individual FIB (*i.e.*, original NH), and which NH is an entry resulting from masking (*i.e.*, completed NH).

When an update occurs in a node of the trie, some completed NHs in the sub-trie rooted at that node will have to be updated to reflect correct matching. This downward propagation in the trie, which we call "masking prefix correction", stops as soon as nodes containing original NH entries are encountered. This process ensures that all the masking prefixes (*i.e.*, prefixes that are not present in the individual FIB but are present in the merged FIB) that are direct descendants of a masked prefix (*i.e.*, a prefix present in the individual FIB), all yield the same NH as the masked prefix. This ensures the correct lookup result.

In the following, we show the update process in detail and illustrate the typical update scenarios in Figure 8. Note that we will use NH Ai in FIB 0, and NH Bi in FIB 1, and that the background colour of the NH denotes the entry type, with white for original NHs, and gray for completed NHs. The original merged trie shown in Figure 8(a) is used as the starting point for each update scenario.

When a new prefix is inserted in an individual FIB, three cases are possible, see Figure 8(b). First, when the prefix already exists in the UPS, the corresponding NH is updated and this NH is flagged as original and the masking prefix correction process is applied from the children of the modified node. For example, if <11*, A8> is inserted into the original merged trie, the corresponding entry becomes <11*, [A8, B3]>, and the masking prefix correction is performed on the sub-trie rooted at <11*, [A8, B3]>. Second, when the prefix (*e.g.*, prefix 010*) corresponds to a new leaf node of the merged trie, the corresponding branch of the trie is extended as required by duplicating the closest ancestor node (*e.g.*, the node <0*, [A1, B1]>) as many times as necessary, but with all entries in the NH array flagged as completed, as those trie nodes correspond to prefixes that are not in the UPS. For example, all NHs in node <01*, [A1, B1]> are flagged as completed. The NH array in the new leaf is identical to that in the nearest ancestor node but for the inserted NH flagged as original in the appropriate FIB (the new leaf is thus added to the UPS). Third, when the prefix

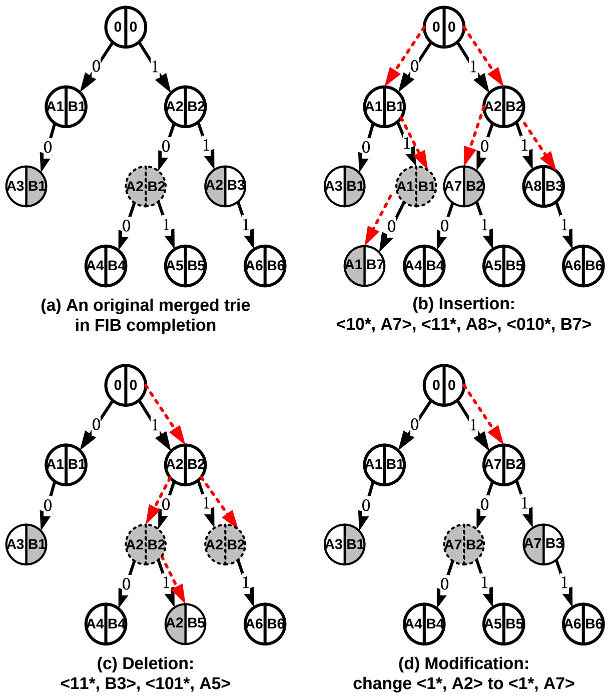

(a) An original merged trie in FIB completion
(b) Insertion: <10*, A7>, <11*, A8>, <010*, B7>
(c) Deletion: <11*, B3>, <101*, A5>
(d) Modification: change <1*, A2> to <1*, A7>

Figure 8: Update scenarios in FIB completion

corresponds to a non-leaf node and this prefix is not in the UPS (*e.g.*, prefix 10*), this prefix is added in the UPS, and the corresponding NH is updated and flagged as original (*e.g.*, <10*, A7>). Besides, the masking prefix correction process is applied on the sub-trie rooted at this node.

When a prefix is deleted from a FIB (*e.g.*, see the deletion of <101*, A5> in Figure 8(c)), the corresponding entry in the NH array in the node where the update occurs is flagged as completed. The NH of this node's closest ancestor (*e.g.*, <10*, A2>) is then determined, and used in the masking prefix correction process applied from the modified node. Note that if the deleted prefix corresponds to the last original entry in the NH array of the node, the prefix will be removed from the UPS. For example, following the deletion of <11*, B3>, prefix 11* will be removed from the UPS, as B3 is the last original NH in the corresponding NH array.

When the NH for an existing prefix is modified, the corresponding NH entry in the appropriate node of the merged trie is also modified, and the masking prefix correction process is applied from the children of the modified node. For example, if we modify <1*, A2> into <1*, A7>, the corresponding NH A2 is replaced by A7. Besides, the completed NHs in the sub-trie rooted at node <1*, [A7, B2]> are updated (*e.g.*, see the NHs in gray associated with prefix 10* and prefix 11* in Figure 8(d)).

Once these update procedures have been applied to the merged trie, all modifications to nodes representing prefixes in the UPS must then be reflected in the TCAM-based lookup engine.

The masking prefix correction process only changes the NHs, not the prefixes, so that for one FIB update, at most one TCAM update should be affected. In the worst case, for one FIB update, the masking prefix correction may traverse the whole trie, and modify a NH in each node. Therefore,

$2^{W+1}-1$ NHs should be modified in the associated SRAM for one update in the worst case, where W is the length of the IP address. We expect the worst case to be unlikely to happen in practice and the average update overhead to be much less costly, as shown in Section 6.

5. FIB SPLITTING

The main drawback of FIB completion is the high worst-case update overhead, which is caused by the masking prefix correction process. Masking prefix correction is required in FIB completion as masking prefixes and their corresponding masked prefix coexist in the same TCAM. A key observation is that, if only disjoint prefixes are merged in the TCAM, the prefix masking issue does not exist any longer, as at most one prefix in a disjoint prefix set can match a given IP address. Previous work [13] has shown that about 90% of prefixes in the tries built from real FIBs are leaf prefixes, which are, by definition, naturally disjoint. This property will also be true for the merged trie built from such FIBs. Based on the above observations, we propose the following approach, which we call *FIB splitting*.

1. The disjoint leaf prefixes in the merged trie are stored in one TCAM based on the basic merged data structure described in Section 3.

2. The remaining prefixes in the merged trie are stored in another TCAM using the non-shared approach.

As disjoint prefixes are merged in their own TCAM, masking prefix correction can be totally avoided. The remaining overlapping prefixes (*i.e.*, they are not necessarily disjoint with each other[1]) [13] are stored in another TCAM, where the prefix masking issue still exists if TCAM FIB merging is performed. However, experiments show that the number of remaining prefixes is very small. Therefore, the non-shared approach is used to manage the remaining prefixes in this second TCAM. By giving priority to matches from the TCAM storing the disjoint leaf prefix set (which necessarily are LPMs), this yields correct lookup results, while keeping good memory scalability as most of prefixes are merged. Additionally, this approach has a more reasonable upper bound on the worst-case update overhead, as masking prefix correction is totally avoided.

5.1 Splitting Approach

In FIB splitting, the first step is to split the prefix set into two sets: a disjoint leaf prefix set and the remaining prefix set (*i.e.*, an overlapping prefix set).

All the FIBs should first be merged, and then the merged prefix set is partitioned. We use the same approach as described in Section 3 for trie merging. Then, the merged trie is split as follows. For a given merged trie, its root node is first checked. If it is a non-leaf node and its corresponding prefix is a valid prefix, this prefix belongs to the overlapping prefix set; if it is a leaf node, its corresponding prefix must be a valid prefix, which belongs to the disjoint leaf prefix set. Then, the left child and right child of the current node are checked recursively.

For example, if the merged trie shown in Figure 7(a) is partitioned, the disjoint leaf prefix set is shown in Figure 9(a). Note that prefixes in this part are merged in one

[1]Prefix overlapping may exist in the remaining set.

prefix	next hop	
00*	A3	0
100*	A4	B4
101*	A5	B5
111*	A6	B6

(a)

prefix	next hop
00*	A1
01*	A2
111*	B3
10*	B1
11*	B2

(b)

Figure 9: (a) The merged disjoint prefix set, and (b) the non-shared overlapping prefix set

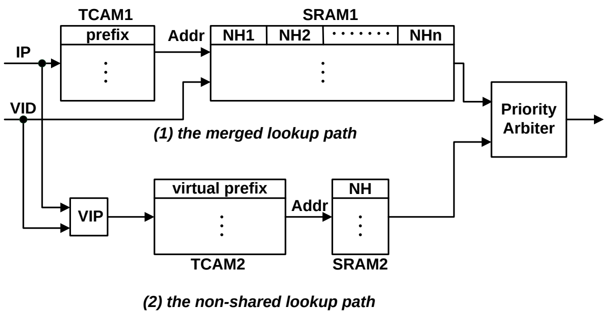


Figure 10: The lookup engine architecture in FIB splitting

TCAM based on the merged data structure shown in Figure 4. The remaining prefixes are overlapping, and we use the non-shared approach to manage them. As in the non-shared approach only prefixes associated with valid NH entries need to be considered, the issue of masking prefixes is avoided altogether in this set, see Figure 9(b).

5.2 Lookup Engine Architecture

The lookup engine architecture is depicted in Figure 10. It consists of two TCAM-based IP lookup paths. The first lookup path (TCAM1/SRAM1) stores the disjoint leaf prefix set (*e.g.*, see Figure 9(a)) as a merged data structure. Note that this disjoint prefix set can be stored in the TCAM without any order constraint. The NH arrays are stored in the associated SRAM. This lookup engine path will be referred to as "*the merged lookup path*".

The second lookup path (TCAM2/SRAM2) stores the overlapping prefix set (*e.g.*, see Figure 9(b)) using the non-shared approach. The virtual overlapping prefix set, which is formed by prepending VIDs to prefixes in the overlapping prefix set, is stored in the TCAM in decreasing prefix-length order. More precisely, only the prefixes with the same VID should be stored in order of relative decreasing prefix lengths. For each entry of the virtual prefix, its associated NH is stored in the SRAM. This lookup engine path will be referred to as "*the non-shared lookup path*".

For an incoming packet, the IP address and the VID are sent to both lookup paths to search in parallel. After both lookups complete, at most two valid NHs are obtained. The priority arbiter module chooses the final NH, based on the observation that if both NHs are valid, the one generated by the merged lookup path always has a higher priority, since the length of the matched prefix in the disjoint prefix set is by design longer than that in the overlapping prefix set.

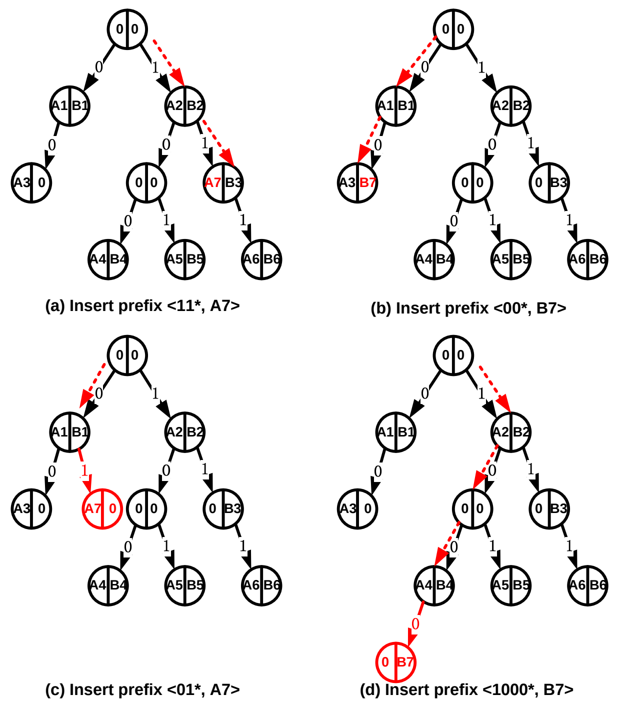

(a) Insert prefix <11*, A7>
(b) Insert prefix <00*, B7>
(c) Insert prefix <01*, A7>
(d) Insert prefix <1000*, B7>

Figure 11: Some insertion scenarios in FIB splitting

5.3 Update Process

One route update involves two phases. First, the update is performed in the auxiliary merged trie in software, and changes in both prefix sets are found; second, according to these changes, update operations are effected in each individual lookup path.

A route update falls into three categories: insertion of a new prefix, deletion of an existing prefix, and modification of an existing prefix. Due to limited space, we only show four typical insertion scenarios in detail. All the other kinds of updates are performed in a similar way. Note, in Figure 11, we will only use NH Ai in FIB 0, and NH Bi in FIB 1, and use the original merged trie shown in Figure 7(a) as the basis from which the updates occur.

Figure 11(a) shows the insertion of a non-leaf prefix 11* with NH A7. This insertion causes the addition of the prefix into the overlapping prefix set for FIB 0. Therefore, virtual prefix 011* and its NH A7 (*i.e.*, entry <011*, A7>) should be inserted in the non-shared lookup path.

Figure 11(b) depicts the insertion of a leaf prefix 00* with NH B7. Prefix 00* already exists in the disjoint prefix set, and thus this insertion only leads to a modification of the corresponding NH array in the SRAM of the merged lookup path. That is, the NH array corresponding to prefix 00* is changed from [A3, 0] to [A3, B7].

Figure 11(c) gives an example of inserting a leaf prefix 01* with NH A7. Since prefix 01* does not already exist in the disjoint prefix set, a new entry <01*, [A7, 0]> should be inserted in the merged lookup path.

Figure 11(d) shows the scenario of inserting a leaf prefix 1000* with NH B7. This insertion turns the prefix 100* from a leaf prefix into a non-leaf prefix, as the prefix 1000* gets inserted as a leaf. As a result, the insertion in this case leads to three changes: (1) entry <1000*, [0, B7]> must be inserted in the merged lookup path; (2) entry <100*, [A4, B4]> should be deleted from the merged lookup path; and (3) entry <0100*, A4> and <1100*, B4> (note that 0100* and 1100* are virtual prefixes) must be inserted in the non-shared lookup path. Changes (1) and (2) can be merged into one write operation by just overwriting entry <100*, [A4, B4]> with entry <1000*, [0, B7]>.

Note that, in software, we do not need to partition the merged trie each time a new route update occurs to find the changes in both prefix sets. Instead, we can find the changes by just performing the update in the merged trie as shown in Figure 11, with time complexity of O(l), where l is the length of the prefix to be updated.

After changes are found in software, update operations must be performed in the lookup paths. We have shown in our previous work [13] that, one route update in the trie always affects at most one leaf prefix and one non-leaf prefix. However, there are some subtle differences in FIB splitting due to the simultaneous existance of multiple FIBs. Changing one leaf prefix leads to at most one write operation in the TCAM of the merged lookup path. Additionally, all the individual NHs associated with this prefix should be written into the associated SRAM. On the other hand, changing one non-leaf prefix triggers the update of at most N virtual prefixes (where N is the number of FIBs) in the TCAM of the non-shared lookup path (*e.g.*, see prefix 100* in Figure 11(d)), as well as N NHs in the associated SRAM.

6. PERFORMANCE EVALUATION

In this section, we evaluate and compare the non-shared approach, FIB completion, and FIB splitting in terms of TCAM size, SRAM size, total cost of the system, and lookup and update performance.

To perform the evaluation, we rely on publicly available BGP routing tables. While these may not be representative of the future routing tables of virtual routers, they provide a reasonable reference point for future research in the area. We collected the 14 full BGP routing tables from the RIPE RIS Project [3], on September 29, 2011. These routing tables were collected from a wide range of locations around the world. In Table 1, we provide the RIPE collector from which each routing table was obtained, the location of the collector, as well as the number of unique IPv4 and IPv6 prefixes in each routing table. The unique prefixes in each routing table are extracted as the prefixes of the corresponding FIB. Note that there are 14 IPv4 FIBs and 13 IPv6 FIBs. In our performance evaluation, we assume that a virtual router platform will host a given number of these FIBs.

6.1 TCAM Size

The TCAM size is determined by the TCAM entry size and the number of entries.

For IPv4 FIBs, the largest prefix length is 32 bits. As we can only get a maximum of 14 IPv4 FIBs for the evaluation, a 4-bit VID is enough to identify them. Therefore, the largest length of a virtual prefix, which is used in the non-shared approach, is 36 bits. Both the lengths of prefix and virtual prefix are under 40 bits, and thus they all fit in TCAMs with 40-bit entry size [2]. As a result, the number of TCAM entries can be used as the only metric to compare TCAM memory requirements in the three approaches.

Figure 12 shows the comparison results of TCAM entry requirements for IPv4 FIBs. In the non-shared approach,

Table 1: Routing tables (2011.09.29, 08:00)

Routing Tables	Location	# of IPv4 prefixes	# of IPv6 Prefixes
rrc00	Amsterdam	399,439	7,218
rrc01	London	375,751	7,294
rrc03	Amsterdam	373,306	7,225
rrc04	Geneva	382,122	5,541
rrc05	Vienna	375,196	7,186
rrc06	Otemachi	367,984	0
rrc07	Stockholm	379,788	7,098
rrc10	Milan	373,024	7,185
rrc11	New York	379,166	7,208
rrc12	Frankfurt	386,924	7,469
rrc13	Moscow	381,561	7,352
rrc14	Palo Alto	380,048	7,280
rrc15	Sao Paulo	392,537	7,137
rrc16	Miami	382,552	6,968

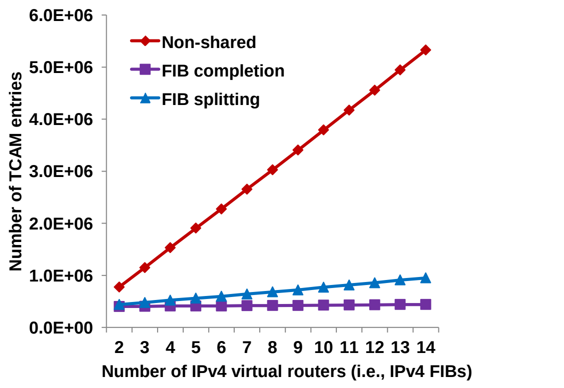


Figure 12: Comparison of TCAM size for IPv4 FIBs

the total number of TCAM entries is the sum of the number of entries in each FIB. Therefore, the TCAM entry requirement increases linearly with the number of virtual routers. When 14 FIBs are stored, 5,329,398 TCAM entries are required in the non-shared approach. In FIB completion, the total number of TCAM entries is equal to the total number of unique prefixes across all FIBs. As the similarity of FIBs is fully exploited (*i.e.*, all the common prefixes are merged in the TCAM), this approach shows the best TCAM memory scalability. With 14 IPv4 FIBs, about 439,467 TCAM entries are required, corresponding to a significant memory reduction of 92% when compared to the non-shared approach. In FIB splitting, most of the prefixes are also merged in the TCAM, and thus good memory scalability is also achieved. For 14 IPv4 FIBs, about 950,722 TCAM entries are required, which corresponds to a memory reduction of 82% when compared to the non-shared approach.

For IPv6 FIBs, the longest prefix is 128 bits. Given a 4-bit VID, the longest virtual prefix is 132 bits. Therefore, the TCAM should be configured to hold 160-bit entries [2] to store the prefixes or virtual prefixes. Again, all prefixes fit in the same TCAM entry size and the number of TCAM entries can be used as the comparison metric. With 13 IPv6 FIBs, the non-shared approach requires 92,161 TCAM entries, while only 7,880 TCAM entries are needed in FIB completion, and 15,783 TCAM entries are needed in FIB splitting, which corresponds to a memory reduction of 91% and 83%, respectively. The growth trends for IPv6 FIBs are similar to those shown in Figure 12, and the proposed two approaches are also more scalable than the non-shared approach in the case of IPv6.

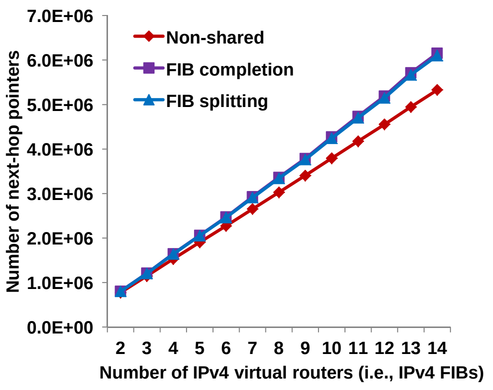


Figure 13: Comparison of SRAM size for IPv4 FIBs

6.2 SRAM Size

Storing NH pointers instead of the complete NHs in the associated SRAM saves memory. Therefore, the number of NH pointers can be used as the metric to evaluate SRAM footprint. In the non-shared approach, the number of NH pointers is equal to the number of TCAM entries. In FIB completion, each prefix is associated with a NH array, which contains N (*i.e.*, the number of FIBs) NH pointers. Therefore, the total number of NH pointers is N times the number of TCAM entries. In FIB splitting, the number of NH pointers in the non-shared lookup path is equal to the number of TCAM entries, and the number of NH pointers in the merged lookup path is N times the number of TCAM entries.

Obviously, as a NH array must be associated with each prefix in the merged FIB, the SRAM usage for FIB completion and FIB splitting are expected greater than that for non-shared approach. This is confirmed in Figure 13, for IPv4 FIBs. When the number of IPv4 FIBs is 14, the non-shared approach only needs to store 5,329,398 NH pointers, against 6,152,538 NH pointers for FIB completion, and 6,103,129 NH pointers for FIB splitting. This inflation of roughly 15% in SRAM footprint is in stark contrast with the reduction of over 80% in TCAM footprint, given the much cheaper cost per MB for SRAMs.

To estimate the corresponding SRAM size, we assume that the NH pointer is 8-bit long, as mentioned in Section 2.1. For 14 IPv4 FIBs, the resulting SRAM size would be 40.7Mb in the non-shared approach, 46.9Mb in FIB completion, and 46.6Mb in FIB splitting. The proposed two approaches thus lead to reasonable SRAM requirements, well below what is available on current line cards. Indeed, the size of a large SRAM on a modern line card is around 72Mb and can go up to 144Mb.

For 13 IPv6 FIBs, the non-shared approach requires 92,161 NH pointers, against 102,440 NH pointers for FIB completion, and 101,991 NH pointers for FIB splitting (about 11% increase). As a result, the SRAM size is 720.0Kb in the non-shared approach, 800.3Kb in FIB completion, and 796.8Kb in FIB splitting.

Table 2: Reference prices of TCAMs and SRAMs

Memory	Part No.	Capacity	Speed	Price
TCAM	NL9512	512K×40bit	250MHz	$387.2
SRAM	CY7C1525	8M×9bit	250MHz	$89.7

Table 3: Cost of the three approaches for IPv4 FIBs

	# of TCAMs	# of SRAMs	Total Cost
Non-shared	11	1	$4348.9
FIB completion	1	1	$476.9
FIB splitting	3	2	$1341

6.3 Total Cost of the System

We now quantify the cost-effectiveness of the trade-off between the reduction in TCAM and the increase in SRAM footprints, as achieved by our proposed FIB merging methods.

To evaluate the cost of the system, we rely on reference prices at the time of writing, see Table 2. The TCAM NL9512 from NetLogic Microsystems contains 512K 40-bit entries, and costs about 390 dollars per chip. The SRAM CY7C1525 from Cypress Semiconductor contains 8M 9-bit entries, and the quoted price is about 90 dollars per chip. Although the actual prices may vary depending on the quantity ordered, the prices shown in Table 2 are representative of the magnitude of the cost.

In our IPv4 scenario, when the number of FIBs is 14, the non-shared approach needs to store 5,329,398 prefixes and 5,329,398 NH pointers, requiring 11 TCAM chips and 1 SRAM chip, for a total cost of about 4350 dollars. In FIB completion, there are 439,467 prefixes and 6,152,538 NH pointers, requiring 1 TCAM and 1 SRAM, for a total cost of about 480 dollars. In FIB splitting, there are two TCAM-based lookup paths. In the merged lookup path, there are 396,339 prefixes and 5,548,746 NH pointers, requiring 1 TCAM and 1 SRAM. In the non-shared lookup path, there are 554,383 prefixes and 554,383 NH pointers, requiring 2 TCAMs and 1 SRAM. Therefore, 3 TCAMs and 2 SRAMs are needed, and the total cost is about 1340 dollars.

Table 3 summarizes the cost of the system for IPv4 FIBs. When compared to the non-shared approach, the total cost can be reduced by 89% in FIB completion, and by 69% in FIB splitting. This result shows that our proposed approaches are cost-effective.

We repeat our cost calculation for the IPv6 scenario. However, the TCAM entry size should be changed to 160 bits for IPv6 prefixes and virtual prefixes. Therefore, the TCAM shown in Table 2 should be configured into the following organization: 128K×160 bit, which is supported by the NL9512 TCAM [2]. Table 4 summarizes the cost of the system for the 13 IPv6 FIBs shown in Table 1.

In the non-shared approach, there are 92,161 prefixes and 92,161 NH pointers, which requires 1 TCAM and 1 SRAM. Therefore, the total cost is about 480 dollars. In FIB completion, there are 7,880 prefixes and 102,440 NH pointers, which requires 1 TCAM and 1 SRAM, for a total cost of about 480 dollars. In FIB splitting, there are 7,184 prefixes and 93,392 NH pointers in the merged lookup path, and 8,599 prefixes and 8,599 NH pointers in the non-shared lookup path. Therefore, 2 TCAMs and 2 SRAMs are needed in FIB splitting, and the total cost is about 950 dollars. FIB splitting costs more than the other two approaches, as it contains two lookup paths, consisting of at least 2 TCAMs and 2 SRAMs irrespective of the size of the prefix sets.

[2]This cost is calculated when using a small TCAM 75P42100.

Table 4: Cost of the three approaches for IPv6 FIBs

	# of TCAMs	# of SRAMs	Total Cost
Non-shared	1	1	$476.9
FIB completion	1	1	$476.9 $189.0[2]
FIB splitting	2	2	$953.8 $378.0[2]

Table 5: Theoretical worst-case lookup performance and update overhead

	Lookup	Update
Non-shared	O(1)	W/2
FIB completion	O(1)	$2^{W+1} - 1$
FIB splitting	O(1)	NW/2

However, it is very important to note that, the TCAM shown in Table 2 is too large for the IPv6 prefix sets in FIB completion and FIB splitting, as the utilization of each TCAM is below 7%. Obviously, smaller and cheaper TCAMs could be used in such a case. For example, the smaller TCAM 75P42100 from NetLogic with 16K 144-bit entries could be used instead. This small TCAM costs only 99.3 dollars per chip. If using this TCAM for the 13 IPv6 FIBs, the cost of FIB completion is about 190 dollars and the cost of FIB splitting is about 380 dollars, both of which are much lower than the cost of the non-shared approach.

6.4 Lookup and Update Performance

We now turn to the lookup and update performance of the three approaches, in terms of theoretical analysis and experimental evaluation.

Table 5 summarizes the theoretical worst-case lookup and update performance. The lookup performance of the three approaches is the same, with O(1) time complexity even in the worst case, thanks to the ability of TCAMs to deliver a deterministic high throughput of one lookup per clock cycle. When using the TCAM and the SRAM shown in Table 2, the lookup throughput can be up to 250 million lookups per second. Considering 64-byte minimum size packets, this lookup rate would sustain well in excess of 100Gbps.

For update performance, we consider the update overhead in the data plane (*i.e.*, in the TCAM-based lookup engines), as updates in the data plane actually affect the packet forwarding process and are therefore critical. Generally, in TCAM-based lookup engines, both the TCAM and its associated SRAM can be updated simultaneously. We assume that, in a TCAM-based lookup engine, a TCAM write operation and an SRAM write operation take the same amount of time, as the TCAM and its associated SRAM are usually driven by the same clock, and both of them usually take just one clock cycle to complete a write operation. In this case, the larger number of memory write accesses in either part can be used as the metric to evaluate the update overhead. Additionally, in the following update evaluation, we refer to the PLO_OPT approach [17] to manage prefixes in the TCAM. That is, when needed, all the prefixes are stored in the TCAM in order of decreasing prefix lengths, and free TCAM space is reserved in the middle of the TCAM.

In the non-shared approach, PLO_OPT requires at most W/2 TCAM memory accesses per route update, where W is the length of the IP address. Therefore, the number of write operations per update is upper bounded by W/2 in the worst case.

In FIB completion, the number of write operations per update is upper bounded by $2^{W+1} - 1$ in the worst case, as one route update may cause at most $2^{W+1} - 1$ NH pointers to be modified in the associated SRAM, due to the masking prefix correction process.

In FIB splitting, one prefix change in the disjoint prefix set requires at most one write operation in the merged lookup path, as it only contains disjoint prefixes and prefix order constraint can be avoided [13]. However, one prefix change in the overlapping prefix set leads to at most N virtual prefix changes in the TCAM of the non-shared lookup path, which causes NW/2 write operations in the worst case when using PLO_OPT [17].

As mentioned before, we expect the worst-case update overhead in our approaches is unlikely to happen in practice. In order to evaluate the actual update overhead, we collected 12 hours' worth of route updates that happened on the 14 IPv4 routing tables shown in Table 1 from the RIPE RIS Project [3]. These update traces contain a total of 22,765,563 prefix announcements and 3,694,904 prefix withdrawals. We replayed these updates in the order of their timestamp, and evaluated the actual update overhead in FIB completion and FIB splitting.

We present in Figure 14, the complementary cumulative distribution of the update overhead over the 12 hours, in FIB completion (Figure 14(a)) and FIB splitting (Figure 14(b)). We observe that in both FIB completion and FIB splitting, most of route updates cost only 1 write access per update, and route updates rarely lead to large number of write accesses in the TCAM-based lookup engines. For example, the percentage of updates costing more than 100 write accesses per update (we call these updates the big updates) is only 0.004% in FIB completion, and only 0.003% in FIB splitting. To fix ideas, we take an approximate calculation: one big update happens roughly every 40s on average (0.004% of updates over 12 hours). If we assume only big updates cause significant packet drop, and every big update needs 100 write accesses (*i.e.*, 400ns total for the chips we consider), the total amount of dropped data is about 5KB, out of a possible 500GB per 40s in a 100Gbps network. The disruption caused by big updates is thus negligible.

Table 6 summarizes the actual update overhead. In FIB completion, the average update overhead in practice is only 1.22 write accesses per update, which is very small and close to the minimum update overhead. This is expected, as we find by experiment, that most updates don't cause any prefix changes in the UPS, and the masking prefix correction often stops quickly in practice. The worst-case update overhead in FIB completion is 2,206 write accesses per update, which is much smaller than what is predicted by the theoretical worst case. In FIB splitting, the average update overhead is only 1.05 write access per update, and the worst-case update overhead is 112 write accesses per update.

Table 6: Update overhead in practice

	Maximum	Average	Minimum
FIB completion	2206	1.22	1
FIB splitting	112	1.05	1

We also evaluate the pre-processing overhead of the updates in software, that does not affect the packet forwarding process. We find by experiment that the average number of trie node accesses per update is about 23 in FIB completion, and about 22 in FIB splitting. We assume that one node access requires one memory reference, which takes 60ns with a cache miss. In this case, we can pre-process over 700K updates per second in software, which exceeds largely the peak update frequency of morden routers [1].

7. RELATED WORK

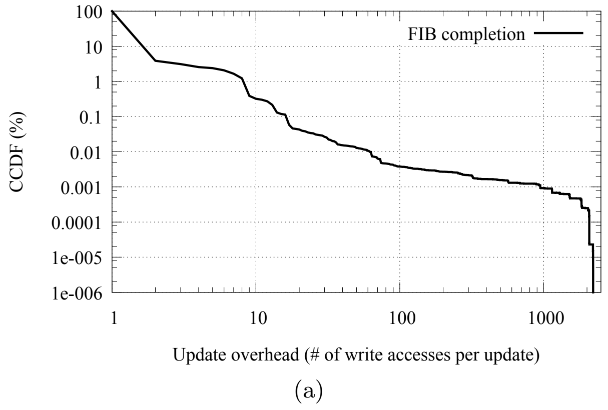
The use of TCAMs to build high-speed lookup engines has been investigated in the past. Previous work mainly focused on addressing the challenges of memory consumption [11,14, 15], power dissipation [23], and update overhead [17,21], in the context of traditional routers, not virtual routers.

In the context of virtual routers, memory scalability is one of the major challenges. Previous work mainly focused on the scalability of SRAM-based virtual routers. Fu *et al.* [7] propose a small, shared data structure to merge multiple tries built from FIBs of virtual routers. The similarity between multiple tries is exploited and many trie nodes are shared during merging, leading to a significantly smaller trie. However, this scheme works well only when the original tries are similar. To address this issue, Song *et al.* [18] propose trie braiding, which enables each trie node to swap its left and right sub-tries freely. Hence, the shape of dissimilar tries can be adjusted and these tries can become as similar as possible. Ganegedara *et al.* [8] argue that, for provider edge router virtualization, even the trie braiding approach is not sufficient since prefix sets belonging to different provider edge routers have different common portions. Therefore, they propose to merge the tries of different provider virtual routers at the split nodes instead of the root nodes. Le *et al.* [10] propose a 2-3 tree-based approach to merge multiple FIBs. A unique virtual ID is attached in front of every prefix in the individual FIB, so that all the FIBs can be merged into a single one directly. Then, a trie is built from the merged FIB, and two disjoint prefix sets are generated by partitioning the trie. Finally, two 2-3 tree-based pipelines are built for the two disjoint prefix sets.

Another related area to ours is FIB aggregation [20, 24], which aims to reduce the size of the FIB. In the context of virtual routers, this approach can be used to compress each FIB separately, and then the compressed FIBs can be put together using the non-shared approach.

8. CONCLUSION

In a virtual router platform, our solutions work best when the prefix sets of different FIBs are similar (*i.e.*, they have a substantial share of common prefixes). Nowadays, private IP addresses are widely used in campus and enterprise networks. It is therefore reasonable to expect virtual routers



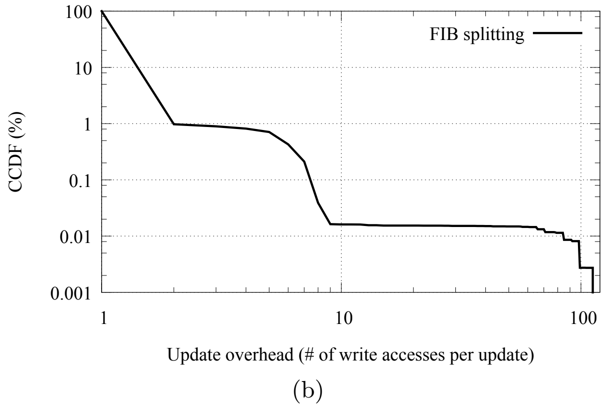


Figure 14: Complementary cumulative distribution function of update overhead in (a) FIB completion and (b) FIB splitting

will likely contain many private addresses in their FIBs, which will have a lot of addresses in common, as the address range of private IP addresses is very limited. In that case, our approaches will be useful.

However, if future virtual routers contain very dissimilar FIBs, merging them directly based on our approaches is not a good idea. For very different FIBs, there will be two possible directions for future work.

First, if dissimilar prefix sets of different FIBs can be transformed into similar ones, our solutions will be useful after the transformation. However, transforming very different prefix sets into similar ones is a major challenge for TCAM-based scalable virtual routers, as it depends largely on the still unknown properties of future FIBs in virtual routers.

Second, if portions of prefix sets are similar, we can merge these portions in TCAMs. If different FIBs do not have a substantial share of common prefixes, but there is similarity among different portions of prefix sets, some solutions, such as multiroot [8], can be adopted in our approaches to merge VPN FIBs in TCAMs. Indeed, multiroot has been proposed to address the dissimilarity problem for VPN FIBs.

However, if the forwarding entries of FIBs are totally different (*e.g.*, name strings for CCN, and IP addresses for IPv4), it is not a good idea to merge these FIBs together using our approaches, as there is little similarity to exploit. A more viable way is to partition all these FIBs into several groups, each group containing the same kind of forwarding entries. For example, one group is for IPv4 FIBs, one group is for IPv6 FIBs, and another group is for CCN FIBs. Then, similarity can be exploited, and merging is performed in each individual group, respectively.

Nevertheless, our proposed approaches have been shown to be very effective when the various FIBs exhibit enough similarity:

1. For large or densely populated FIBs, such as in the case of IPv4, the proposed two approaches exhibit very good scalability. Indeed, the TCAM size requirements are much lower, at the cost of marginally larger SRAM requirements, which is very cost-effective.

2. For small or sparsely populated FIBs, as currently in IPv6, our approaches also exhibit very good scalability. However, if FIB size is not so important, the non-shared approach might be a good choice as it guarantees a low upper bound on the update overhead.

3. Both FIB completion and FIB splitting perform well in terms of lookup and update performance. The lookup performance is up to 1 lookup per clock cycle, and the average update overhead is as low as around 1 write access per update. Although the theoretical worst-case update overhead is high, it is a loose bound unlikely to happen in practice.

In many practical settings, our proposed merged data structure is therefore effective at reducing TCAM footprint in virtual routers, and our proposed algorithms are efficient, at resolving the prefix masking issue encountered in TCAM FIB merging.

As far as we know, our work is the first to exploit the possibility of using TCAMs to build scalable virtual routers. In future work, we will implement our approaches on our PEARL platform [22], and address the remaining challenges, such as power consumption and dissimilar FIBs.

9. ACKNOWLEDGMENTS

We thank our shepherd Patrick Crowley and the anonymous reviewers for their valuable comments on the paper.

This work was supported in part by National Basic Research Program of China with Grant 2012CB315801, National Natural Science Foundation of China (NSFC) with Grants 61133015 and 61061130562, Strategic Priority Research Program of the Chinese Academy of Sciences with Grant XDA06010303, and National Science and Technology Major Project of Ministry of Science and Technology of China with Grant 2011ZX03002-005-02.

10. REFERENCES

[1] BGP Reports. `http://bgp.potaroo.net/`.
[2] NetLogic NL9000 Family. `http://www.netlogicmicro.com/Products/Layer4/index.asp`.
[3] RIPE RIS Raw Data. `http://www.ripe.net/data-tools/stats/ris/ris-raw-data`.
[4] M. B. Anwer, M. Motiwala, M. b. Tariq, and N. Feamster. Switchblade: a platform for rapid

deployment of network protocols on programmable hardware. In *Proc. ACM SIGCOMM*, pages 183–194, 2010.

[5] N. M. M. K. Chowdhury and R. Boutaba. A survey of network virtualization. *Computer Networks*, 54(5):862–876, 2010.

[6] N. Egi, A. Greenhalgh, M. Handley, M. Hoerdt, F. Huici, and L. Mathy. Towards high performance virtual routers on commodity hardware. In *Proc. ACM CoNEXT*, pages 1–12, 2008.

[7] J. Fu and J. Rexford. Efficient IP-address lookup with a shared forwarding table for multiple virtual routers. In *Proc. ACM CoNEXT*, pages 1–12, 2008.

[8] T. Ganegedara, W. Jiang, and V. Prasanna. Multiroot: Towards memory-efficient router virtualization. In *Proc. IEEE ICC*, pages 1–5, 2011.

[9] T. Ganegedara, H. Le, and V. K. Prasanna. Towards on-the-fly incremental updates for virtualized routers on FPGA. In *Proc. International Conference on Field Programmable Logic and Applications (FPL)*, pages 213–218, 2011.

[10] H. Le, T. Ganegedara, and V. K. Prasanna. Memory-efficient and scalable virtual routers using FPGA. In *Proc. ACM/SIGDA FPGA*, pages 257–266, 2011.

[11] H. Liu. Routing table compaction in ternary CAM. *IEEE Micro*, 22(1):58–64, 2002.

[12] W. Lu and S. Sahni. Low-power TCAMs for very large forwarding tables. *IEEE/ACM Transactions on Networking*, 18(3):948–959, 2010.

[13] L. Luo, G. Xie, Y. Xie, L. Mathy, and K. Salamatian. A hybrid IP lookup architecture with fast updates. In *Proc. IEEE INFOCOM*, pages 2435–2443, 2012.

[14] V. Ravikumar and R. N. Mahapatra. TCAM architecture for IP lookup using prefix properties. *IEEE Micro*, 24(2):60–69, 2004.

[15] V. Ravikumar, R. N. Mahapatra, and L. N. Bhuyan. EaseCAM: an energy and storage efficient TCAM-based router architecture for IP lookup. *IEEE Transactions on Computers*, 54(5):521–533, 2005.

[16] M. A. Ruiz-Sanchez, E. W. Biersack, and W. Dabbous. Survey and taxonomy of IP address lookup algorithms. *IEEE Network*, 15(2):8–23, 2001.

[17] D. Shah and P. Gupta. Fast updating algorithms for TCAMs. *IEEE Micro*, 21(1):36–47, 2001.

[18] H. Song, M. Kodialam, F. Hao, and T. Lakshman. Building scalable virtual routers with trie braiding. In *Proc. IEEE INFOCOM*, pages 1–9, 2010.

[19] J. S. Turner, P. Crowley, J. DeHart, A. Freestone, B. Heller, F. Kuhns, S. Kumar, J. Lockwood, J. Lu, M. Wilson, C. Wiseman, and D. Zar. Supercharging planetlab: a high performance, multi-application, overlay network platform. In *Proc. ACM SIGCOMM*, pages 85–96, 2007.

[20] 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 *Proc. ACM CoNEXT*, pages 1–12, 2011.

[21] B. Vamanan and T. N. Vijaykumar. TreeCAM: decoupling updates and lookups in packet classification. In *Proc. ACM CoNEXT*, pages 1–12, 2011.

[22] G. Xie, P. He, H. Guan, Z. Li, Y. Xie, L. Luo, J. Zhang, Y. Wang, and K. Salamatian. PEARL: a programmable virtual router platform. *IEEE Communications Magazine*, 49(7):71–77, 2011.

[23] F. Zane, G. Narlikar, and A. Basu. CoolCAMs: power-efficient TCAMs for forwarding engines. In *Proc. IEEE INFOCOM*, pages 42–52 vol.1, 2003.

[24] X. Zhao, Y. Liu, L. Wang, and B. Zhang. On the aggregatability of router forwarding tables. In *Proc. IEEE INFOCOM*, 2010.

[25] K. Zheng, C. Hu, H. Lu, and B. Liu. A TCAM-based distributed parallel IP lookup scheme and performance analysis. *IEEE/ACM Transactions on Networking*, 14(4):863–875, 2006.