This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the IEEE ICC 2011 proceedings

Multiroot: Towards Memory-Efficient Router Virtualization

Thilan Ganegedara, Weirong Jiang, Viktor Prasanna
University of Southern California
3740 McClintock Ave., Los Angeles, CA 90089
Email: {ganegeda, weirongj, prasanna}@usc.edu

***Abstract*—Router virtualization has become a powerful technique to make efficient use of networking hardware. It allows multiple virtual networks to co-exist on the same physical networking substrate. This requires the hardware router to maintain multiple lookup tables. Hence, ultimately the hardware router should be capable of handling packets from different virtual networks. In this paper, we introduce a memory-efficient solution for router virtualization named, *Multiroot*. We propose this potential scheme, for Provider Edge (PE) router virtualization after examining the address space allocation of such networks. Multiroot is a novel merging technique to consolidate all the routing tables to a single merged table. The shared data structure used in our algorithm results in a significant memory usage reduction in the lookup data structure while guaranteeing traffic isolation which is critical in a virtualized environment. This improvement in memory usage results in a very scalable solution for router virtualization in terms of resource usage of the hardware router. Multiroot uses trie data structure and can be implemented on a hardware or a software platform. Experiments show that our solution can achieve up to 5 fold memory usage reduction compared to state-of-the-art techniques present in literature.**

I. Introduction

Network virtualization [1] was introduced to overcome the deficiencies inherent in traditional networks, such as underutilization, protocol rigidity, etc. It allows the Internet Service Providers (ISPs) to define multiple virtual networks on top of the physical network so that the underutilized networking hardware can be efficiently used to accommodate multiple networks. This makes device consolidation possible, which results in a significant saving in terms of the cost of networking devices, power consumption, maintenance cost etc. Further, each virtual network may run different protocols for packet forwarding which increases the flexibility in the network [1].

An example virtual networking topology is illustrated in Figure 1. Virtual networks A and B are created on top of the physical network which does the actual routing of packets. In the virtual domain, packet routing is considered independent of the underlying physical network. Moreover, the virtual networks A and B may run different protocols for packet routing in the virtual domain. Hence network virtualization is considered a more flexible solution for packet routing compared to the existing protocol-bound networking devices.

At the physical layer, a virtualized router has to maintain multiple routing tables to serve traffic from multiple networks. This is known as router virtualizaition. Each virtual routing table represents a particular virtual network and the routing table size is clearly determined by the size of the network. Provider edge (PE) networks are relatively small compared to core networks. For example, a provider edge routing table contains 1k-10k prefixes [9] while a core routing table, in average, has over 1M prefixes [10].

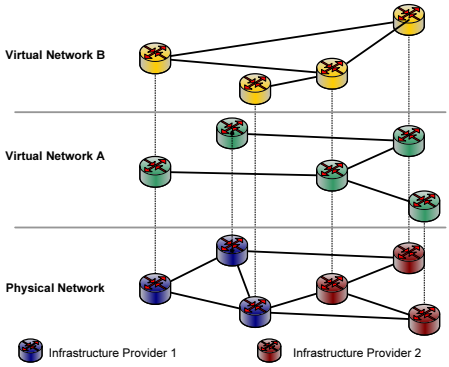

Fig. 1. Example virtual network topology

The main limitation in virtualizing a router is, scalability. By scalability, we refer to the number of virtual networks supported on the available hardware resources. Unless there are abundant resources, techniques to reduce the resource requirement should be considered in order to improve the scalability of the virtualized router. In this paper we present *Multiroot*, a memory-efficient and scalable solution for router virtualization. Multiroot supports all the features of network virtualization such as security, traffic isolation, etc. By observing the IP address space allocation for different networks [12], we propose an efficient merging technique, in order to achieve a significant memory usage reduction which is a critical limiting component in such virtualized environments. Our main contributions in this paper are:

- Novel merging technique: To exploit the structure of different virtual routing tables
- Memory-efficiency: Up to 5 fold memory usage reduction compared to state-of-the-art
- Improved scalability: Accommodate multiple virtual networks without exploding the memory requirement
- Shared data structure: Guarantees traffic isolation among virtual networks

The rest of the paper is organized as follows: Section II discusses related work; Section III introduces our Multiroot algorithm; Section IV presents the experimental results of the proposed algorithm; Section V concludes our paper.

II. Related Work

Network virtualization has recently attracted a lot of interest from the networking industry as well as from research community. In [1], the authors present a comprehensive study on network virtualization in terms of its advantages, design goals and possible challenges that might arise. Despite its advantages, little work has been done in terms of algorithmic/architectural contributions to support network virtualization on the physical networking substrate.

Two approaches for router virtualization exist in the literature. One is the *merged* approach in which all the routing tables are merged into a single routing table. The other one is *separated* in which there is a separate router instance for each virtual network. Each approach has its own advantages and disadvantages and can be used depending on the requirement and available resources.

Several networking device manufacturers have introduced network virtualization on their routers [7], [8] to support up to hundreds of virtual networks. Cisco [7] proposes a software and hardware virtualized router and Juniper [8] achieves router virtualization by instantiating multiple router instances on a single hardware router to enforce security and isolation among virtual routers.

In [2] the authors present a memory-efficient data structure for IP lookup in a virtualized router. They take the merged approach and achieve significant memory saving by using a shared data structure. Their algorithm performs well when the routing tables have similar structure. Otherwise, the memory requirement increases significantly. Song et al. in [5] takes a different approach for the *merged* router virtualization problem by mapping the tries of the routing tables to a merged trie, to increase the overlap among the different tries. For this they introduce *braiding bits* at each node which increases the complexity. Even though memory efficiency is claimed, the complexity of this algorithm makes it less appealing to real networking environments.

In [6] authors take the *separated* approach in which they implement multiple virtual router instances on hardware (on NetFPGA [11]) and on software (running on a virtualized general purpose computer). They show the scalability of their design for up to 16 routing tables. However, the throughput of their overall router is relatively low ($\sim$100Mbps) and the scalability is limited due to extensive hardware resource usage.

In this paper, we propose a potential scheme to realize scalable router virtualization, that results in a highly memory efficient solution. This method exploits the address space allocation of provider edge networks to further reduce the memory requirement of the router which improves the scalability of our virtualized router significantly. We take the merged approach for router virtualization by using a shared trie data structure.

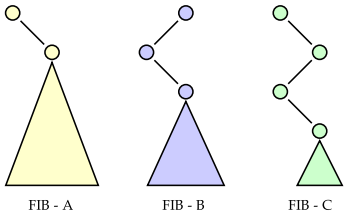

Fig. 2. FIBs with different prefix values/lengths

III. Our Solution

In this section we introduce our algorithm and the shared data structure used for router virtualization. In Section IV we show that, by using our scheme, the scalability of the virtualized router can be improved significantly.

A. Virtual Router Address Space

As mentioned in Section I, edge networks are relatively small compared to core networks. And they are defined in a specific range of addresses assigned by the ISP [12]. Therefore, packet forwarding in such networks is performed within a small range in the address space of either IPv4 or IPv6. Hence, it is possible to define the complete Forwarding Information Base (FIB) for an edge router in a sub-space of the complete address space. Connection to the Internet and other networks is provided using gateway routers. Therefore, within the edge network, IP addresses are limited to a specific range. This observation led us to the idea of Multiroot in which we exploit this feature of PE routing tables.

When such a network is mapped on to a trie, the range of IP addresses are located at a specific branch of the trie starting at the root. This is illustrated in Figure 2 where the FIBs corresponding to the three edge networks `A`,`B` and `C` are mapped on to three binary tries. It becomes obvious that the prefixes in a particular edge network have a common portion. We call this common portion as *common prefix*. In Table I, we list the common prefix information and address space corresponding to the three FIBs in Figure 2.

The structure and the size of the PE routing tables can be significantly different from one another. It depends on the address range allocated to that network and how the network is configured by the network administrator. To realize router virtualization, such different routing tables should be merged in a way that does not exponentially increase the router's hardware resource requirement. Otherwise, the scalability of the hardware router becomes severely limited when more and more routing tables are merged. In the following section, we describe how to overcome these scalability issues by using a simple, yet a powerful algorithm to merge different routing tables.

B. Motivating Example

We use a simple example to show the advantages of our scheme compared to the other existing schemes. Suppose that we have two routing tables to merge, FIB-A and FIB-B in Figure 2. These two routing tables reside in the two branches

TABLE I
COMMON PREFIX INFORMATION FOR THE FIBS IN FIGURE 2

FIB ID	Common Prefix	Prefix Length	Address Space
A	1*	1	2 Billion
B	01*	2	1 Billion
C	101*	3	512 Million

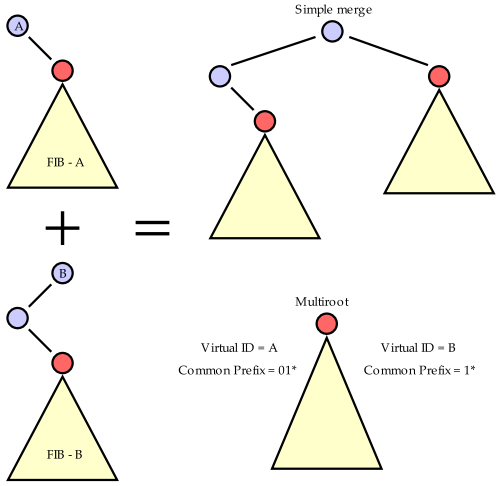

Fig. 3. Merging process: Simple merging vs. Multiroot

of the trie starting from the root. FIB-A on the right sub-trie and FIB B on the left sub-trie. For simplicity, let us assume that we merge FIB-B on to FIB-A.

Merging them as proposed in [2] will result in creating a new set of leaf and non-leaf nodes corresponding to FIB-B which will result in a merged trie with a memory requirement greater than the sum of the memory requirements of the two tries. The memory requirement becomes greater because now each leaf node has to store next hop information for two FIBs. Trie braiding [5], will result in maximum overlap between the two tries, but the complexity and memory requirement at each node increases, because all the non-leaf nodes have to store braiding bits for each routing table.

The advantage of our algorithm is highlighted in Figure 3. The amount of overlap is much higher than that of [2] since we merge at the split nodes. Also we do not require any extra information to traverse the trie. Consequently, our approach requires less amount of memory compared to [2] and [5].

C. Merging Algorithm

In order to take advantage of the common prefix mentioned in Section III-A, we use an effective merging technique to reduce the overall memory requirement and the lookup complexity of the resulting router. Initially, we build a uni-bit trie for each virtual FIBs. Then we execute a recursive algorithm to find the node at which the uni-bit trie starts its first split. For example, the trie corresponding to FIB B in Figure 2, starts to split at the sub-trie starting at 01*. The very first node to have two children is considered the *split node*. All the previous nodes have only a single child. In Algorithm 1 we describe the our algorithm to find the split node of the constructed trie.

Algorithm 1 FindSplit(node n, string prefix)

Require: node n, string *prefix*

```
1: if n → left == 0 and n → right != 0 then
2:     prefix.push_back('1')
3:     return FindSplit(n → right, prefix)
4: else if n → left != 0 and n → right == 0 then
5:     prefix.push_back('0')
6:     return FindSplit(n → left, prefix)
7: else
8:     return prefix
9: end if
```

After the split node is found, we truncate the constructed uni-bit trie at the split node. Then, we merge this truncated trie on to a *merged* trie, where the merged trie is the trie that holds routing information for all the considered virtual networks (initially, the merged trie is an empty trie). The same process is repeated for all the other tries and the merged trie is augmented. Once all the FIBs are merged this way, we do a leaf push [4] to bring all the forwarding information down to the leaf nodes.

In addition to the above process, the information about the split nodes of all the tries are stored as a lookup table similar to the table shown in Table I, at the root node. This table is used for the initial lookup. When a packet arrives, the virtual ID will be used to access the table and the it will output the necessary information to begin the IP lookup process.

In the original trie for a given virtual routing table, we might encounter a case where there might be an IP-prefix above the split node. For example, the default gateway of a router is specified for the prefix 0.0.0.0/0 (default gateway) so that if there is no match for an incoming packet, this information will be used for packet forwarding. If such a case occurs, we can store that next hop information in a lookup table similar to Table I. This information can be carried along with the packet and if there is no other match, the default next hop information can be used to route the packet.

Note that for a set of core routing tables, our algorithm will perform exactly like [2]. The reason for this is that, for a core routing table, the IP address range is fairly wide, and it might span over the whole IP address range. Hence the root node will become the split node, since there will not be any common prefix.

D. Lookup Process

The lookup process in Multiroot is very similar to the existing trie based IP lookup schemes proposed in the literature either on hardware [4] or using software [2]. The major differences in Multiroot appear in the initial lookup stage and when accessing a leaf node. Figure 4 illustrates the lookup process in Multiroot.

When a packet arrives, its destination IP and the Virtual Network Identifier (VNID) are extracted. The VNID is used to access the lookup table shown in Figure 4 to find the

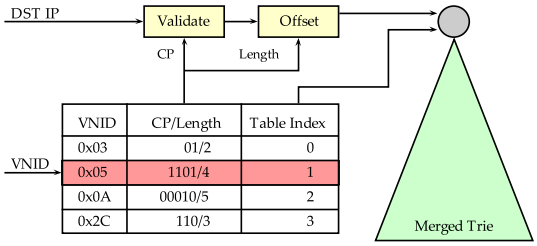

Fig. 4. IP Lookup process in Multiroot

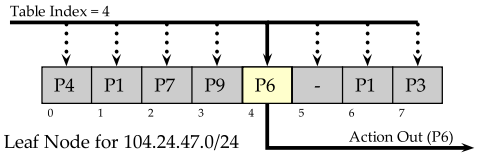

Fig. 5. Shared leaf node structure to serve multiple virtual networks

corresponding common prefix (CP) and the table index. The *table index* is an internal mapping for VNID to simplify the lookup process. When this information is looked up in the table, the destination IP is compared with the common prefix corresponding to that FIB to validate the packet.

In Multiroot, the root of the merged trie might not be the root of the original trie corresponding to the FIB the incoming IP belongs to. Therefore, we cannot start the lookup process at the zeroth bit position of the incoming IP. We use the length of the common prefix to give an offset to the index of the IP where we start the lookup process within the merged trie. This allows us to start the lookup process at the correct bit position of the incoming IP. Since multiple logical roots are mapped on to the root of the merged trie, we name our approach *Multiroot*.

E. Shared Data Structure

In network virtualization, security is of utmost importance. Even though the hardware is shared among multiple virtual routers, packets that belong to a specific virtual network should not interfere with traffic from any other virtual network. The two virtualizing methods described in Section I, *merged* and *separate* implement security in two different ways. The former uses soft isolation using the data structure, whereas the latter uses hard isolation by implementing multiple separate router instances. Since we take the merged approach, we use the leaf node data structure to isolate traffic from different virtual networks.

The non-leaf nodes, which are used to traverse the trie to locate the leaf node, are common to all the virtual networks. Whereas in every leaf node, there is an entry for each virtual FIB. Such an entry contains the forwarding information related to a specific virtual FIB. When a leaf node is reached, the next hop information will be accessed using the table index. In Figure 5, this process is illustrated for the leaf node with prefix 104.24.47.0/24, virtualized for 8 virtual networks.

F. Theoretical Comparison with Existing Approaches

The existing router virtualization methods exhibit similar construction. Table II compares the running time and the

TABLE II
THEORETICAL COMPARISON

Method	Memory Requirement	Time complexity
Simple merging [2]	$\Omega(K * N_{max})$	$O(NlogN)$
Trie braiding [5]	$\Omega(K * N_{max})$	$O(N^2)$
Separate [6]	$O(\sum_{i=0}^{K} N_i)$	$O(KN)$
Multiroot	$\Omega(K * N_{max})$	$O(KN)$

memory requirement of these methods. Here, N is the average number of nodes in a trie and K is the number of virtual routers. N_i and N_{max} are the number of nodes in each trie and maximum trie node count, respectively.

It should be noted that N_{max} for Multiroot is smaller compared to N_{max} of simple merging and trie braiding because of the common prefix extraction. Hence, Multiroot results in a very attractive memory efficiency and a fast execution time which is critical for quick updates.

IV. Experimental Results

The evaluation of our algorithm with respect to the memory requirement and execution time is presented in this section. Specifically, the performance, impact of common prefix length to the overall performance of our algorithm and the execution time are analyzed.

A. Routing Table Sources

We propose Multiroot for PE networks. For these networks, the table size is generally in the range of 1k-100k prefixes [9]. To conduct our experiments, we did not have access to any provider edge routing tables. Also partitioning existing core routing tables results in unrealistic routing tables. To avoid this problem, we used a synthetic rule generator [3] to generate routing tables. This provided us the flexibility to generate close-to-real synthetic routing tables with different structures and different common prefixes.

We generated 16 routing tables, each having 100k prefixes to illustrate the performance of our algorithm for a worst case scenario in PE networks. These tables were generated in such a way that each routing table followed the structure of one of the core routing tables listed in [10].

B. Memory-Efficiency and Scalability

This experiment was conducted using routing tables with random common prefixes. The prefix lengths ranged from 2 to 5. Figure 6a illustrates the total number of leaf nodes in the merged trie. It shows that as we increase the number of routing tables, the total number of nodes start to saturate, because the trie becomes dense and complete. Therefore, adding more tries does not result in adding more nodes.

Figure 6b depicts the total memory requirement of the merged trie including the leaf and non-leaf nodes. In our analysis we considered each leaf node to be of 32-bit size (2*16-bit pointers) and the size of a leaf node is calculated as `#of leaf nodes*K*6` where `K` is the number of virtual networks and 6 is the bit width of each next hop field.

These results show that our solution requires much less memory compared with the existing methods [2], [6]. We

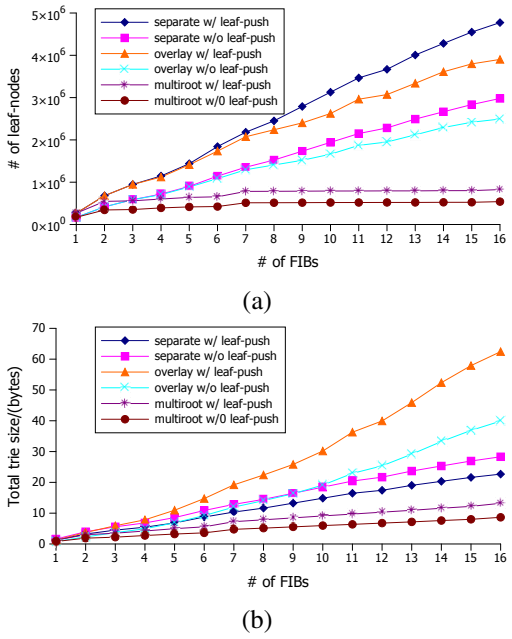

Fig. 6. Node and memory analysis for 16 virtual routing tables

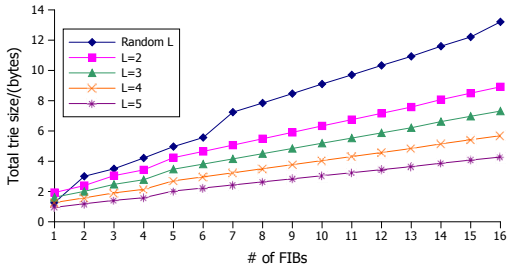

Fig. 7. Memory requirement variation for different common prefix lengths

achieve 6 fold and 5 fold memory reduction compared with the separate router approach [6] and simple overlaying [2] respectively. Therefore, our solution outperforms existing methods in terms of the total number of nodes and most importantly, in terms of the total memory requirement. This makes Multiroot a scalable solution for router virtualization with respect to router's hardware resource usage.

C. Constant length common prefix

Figure 6 illustrates memory savings we achieve in the case of random common prefix. Figure 7 illustrates the memory requirement for the random length common prefix case and for controlled common prefix case. `L` corresponds to the length of the common prefix. As can be seen, the memory requirement of the merged trie can be further reduced significantly by merging the routing tables with the same length common prefix.

D. Execution Time of Multiroot

In network virtualization, addition of a new virtual network to the existing virtualized router should be done fairly quickly. Further, prefix updates should be performed in a quick fashion. Therefore the time overhead involved in these operations should be as small as possible.

As shown in Table II, Multiroot has a decent time complexity compared to the existing approaches. The simplicity of our merging process increases the speed of solution so that the time taken to reconstruct the merged routing table can be reduced. Even though our experiments were conducted for fairly large routing tables (compared to the typical edge routing tables), for the 16 routing tables mentioned in Section IV-A our algorithm completed the merging process in 3.2s on a Quad-core AMD Opteron processor running at 2.00GHz.

V. Conclusion

In this paper, we presented Multiroot, a novel approach for router virtualization. We propose this scheme by observing the address space allocation for edge networks and by exploiting the structure of such routing tables. Due to the reduced memory consumption of our solution, we have improved the scalability of our solution considerably, in terms of hardware resource requirement. The shared data structure used in our architecture guarantees traffic isolation which is critical in network virtualization. Multiroot can be used as a hardware or software solution for IP lookup in virtualized routers.

We plan to extend Multiroot for IPv6 router virtualization where prefixes are explicitly used in the address. For such routing tables, Multiroot becomes a very appealing solution. Further, the virtual routing tables can be grouped and merged by considering the length of the common prefix, size and the node count as metrics to improve the performance of our algorithm.

References

[1] N.M. Chowdhury, Kabir Mosharaf and Raouf Boutaba, "A Survey of Network Virtualization," The International Journal of Computer and Telecommunications Networking, 2010, pp. 862 - 876.
[2] Jing Fu and Jennifer Rexford, "Efficient IP-address lookup with a shared forwarding table for multiple virtual routers," in proc. ACM CoNEXT, 2008, pp. 1-12
[3] T. Ganegedara, W. Jiang, V.K. Prasanna, "FRuG: A Benchmark for Packet Forwarding in Future Networks," to appear in proc. IEEE IPCCC 2010.
[4] H. Le, W. Jiang, V.K. Prasanna, "A SRAM-based architecture for Trie-based IP Lookup using FPGA", in proc. IEEE FCCM 2010, pp. 33-42.
[5] Haoyu Song, M. Kodialam, Fang Hao, T.V. Lakshman, "Building Scalable Virtual Routers with Trie Braiding," in proc. IEEE INFOCOM, 2010, pp. 1-9
[6] Deepak Unnikrishnan and Ramakrishna Vadlamani, Yong Liao and Abhishek Dwaraki Jeremie Crenne, Lixin Gao, Russell Tessier, "Scalable network virtualization using FPGAs", in proc. ACM/SIGDA FPGA, 2010, pp. 219 - 228
[7] Router virtualization in service providers. Available from: http://www.cisco.com/en/US/solutions/collateral/ns341/ns524/ns562/ns573/white_paper_c11-512753.pdf.
[8] Control plane scaling and router virtualization. Available from: http://www.juniper.net/us/en/local/pdf/whitepapers/2000261-en.pdf.
[9] Edge routing table archives. Available from: http://bgp.potaroo.net/index-bgp.html.
[10] Core routing table archives. Available from: http://www.ripe.net/ris/.
[11] NetFPGA. Available from: http://netfpga.org/.
[12] IPv4 Address Space Registry. Available from: http://www.iana.org/assignments/ipv4-address-space/ipv4-address-space.xml.