ISSN:2319-9873

Research & Reviews: Journal of Engineering and Technology

Scalable and Efficient Tree based Memory Mapping in Virtual Routers

Deepika V[1]*, Geetha SK[1] and Varunkumarr CV[2]

[1]Assistant Professor, Department of CSE, RVS College of Engineering and Technology, Anna University, Coimbatore, India

[2]System Engineer, Infosys Limited, Chennai, India

Research Article

Received date: 17/07/2015

Accepted date: 24/09/2015

Published date: 30/09/2015

***For Correspondence**

Deepika V, Assistant Professor, Department. of CSE, RVS College of Engineering and Technology, Anna University, Coimbatore, India
E-mail: iamvdeepika@gmail.com

Keywords Dynamic Programming, Trie Braiding, Virtual Router.

ABSTRACT

Many well liked algorithms for fast packet forwarding and filtering rely on the tree facts and figures structure. Examples are the triebased IP lookup and package classification algorithms. With the latest interest in mesh virtualization, the ability to run multiple virtual router examples on a common personal router stage is absolutely vital. A significant climbing issue is the number of virtual router examples that can run on the platform. One limiting factor is the allowance of high-speed recollection and caches available for storing the packet forwarding and filtering data organizations. A perfect goal is to accomplish good climbing while sustaining total isolation amidst the virtual routers. Although total isolation requires sustaining distinct facts and figures structures in high speed memory for each virtual router. In this paper, we study the case where some sharing of the forwarding and filtering facts and figures organizations is permissible and evolve algorithms for blending tries used for IP lookup and packet classification expressly, we evolve a means called trie braiding that allows us to blend tries from the facts and figures organizations of different virtual routers into just one compact trie. Two optimal braiding algorithms and a faster heuristic algorithm are offered, and the effectiveness is demonstrated utilizing the real-world facts and figures sets.

INTRODUCTION

The growing demand for easy mesh customization and discovery has led to expanded interest in building virtual systems [1–4]. Establishing these virtual networks needs the use of virtual routers that proceed individually of sharing widespread physical resources. An example of the need for differing routing purposes on the same stage, akin to virtualization, is the case of ipv4-to-ipv6 migration [5,6]. Since the migration can only occur incrementally, the so-called dualstack router needs to support both ipv4 and ipv6 routing simultaneously. In each such router, packets are classified based on the IP type and then forwarded utilizing the corresponding forwarding benches [7].This is alike to having two virtual routers on the same stage.

As the need of virtualized networks augments, a personal router is anticipated to support a few tens and probably even hundreds of virtual routers, with each having its own forwarding table and routing protocol examples (e.g., juniper routers currently are reported to be capable of carrying 16 virtual routers [8]). Scaling to these figures poses serious trials to the router conceive, particularly when complex package classification and deep package examination need to be implemented.

A critical resource that limits scalability is the amount of high-speed memory (e.g., sram and tcam) used for caching the packet forwarding and filtering data structures, which account for a large portion of the system cost and power consumption. even though it is straightforward to partition the memory and allocate to each virtual router separately, this is not efficient for two

reasons: 1) it is difficult to determine the right fractions to allocate to each virtual router when each virtual router has different forwarding table size and that can change dynamically; 2) overall memory consumption is linear in the number of virtual routers as well as in the size of the memory required for each virtual router. For example, the latest bgp table already contains about 350 k prefixes [9]. meanwhile, a state-of-art 18-mb tcam can only store 500 k ipv4 (or 250 k ipv6) prefixes, which is hardly sufficient for two unique bgp tables.

If the isolation obligation is calm, recollection usage can be considerably reduced by consolidating the one-by-one package forwarding and filtering facts and figures structures into a combined one. we display the advantages of this idea by evolving effective algorithms for blending the tree data organisations needed for accomplishing two

key router functions: longest prefix matching (lpm) and packet classification.

A scheme for reducing the allowance of recollection used for lpm in virtual routers in offered in [9]. In that work, the tries sustained by each virtual router for longest prefix matches are blended into one trie by exploiting prefix commonality and overlaps between virtual routers. the recollection decrease profited through the overlap is largely due to the likeness between the tries. however, when the tries are functionally distinct, the gains utilising this design are diminishing.

We insert a new means called trie braiding that can be utilised to construct a compact distributed facts and figures structure. Trie braiding enables each trie node to swap its left child node and right progeny node without coercion. The changed form is memorized by a lone bit at each trie node. this additional degree of freedom devotes us the opportunity to accomplish the optimal recollection distributing presentation when blending the endeavours. We develop two optimal dynamic programming algorithms for blending multiple trees into a lone compact tree, as well as a much quicker heuristic algorithm. Trie braiding directs to important savings in high-speed recollection required for classification and longest-prefix matching, and hence advances scalability.

BACKGROUND

Trie-Based Lpm In Virtual Routers

The trie-based LPM algorithms start from a simple binary trie formed by the IP prefixes. Each binary trie node has two pointers pointing to its two progeny nodes, “0” on the left and “1” on the right. When injecting a prefix into the trie, we analyze the prefix morsels in alignment and use the bit worth to walk the binary trie. We pursue the left pointer if the bit value is “0,” and the right pointer else. New nodes are created dynamically. This method is preceded until all the prefix bits are consumed. The last trie node is then labeled as a legitimate prefix those aides with a next-hop. As an example, two forwarding benches are shown in **Table 1.**The corresponding binary tries are shown in **Figure 1.**

Table 1. Sample prefix tables.

Table A	Table B
0*	0*
1*	110*
01*	111*
001*	

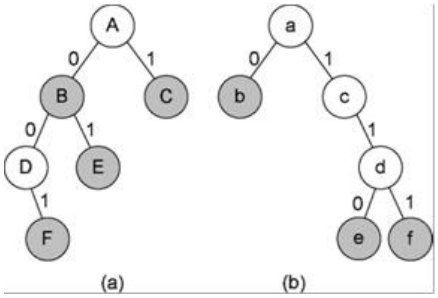

Figure 1. Binary tries for the sample prefix tables. (a)For Table A. (b)For Table B. The dark nodes are valid prefix nodes.

The lookup method utilizes the packet’s place visited IP address to traverse the trie. The worth of the current address bit determines which trie agency is to be taken. During the traversal, the last matching prefix node along the route is recorded as the current best agree. When the route finishes, the lookup process returns the noted prefix as the best matching prefix. Its associated next-hop is then utilised to ahead the packet.

Problem Statement

When multiple virtual routers are hosted on the identical physical router, with each virtual router requiring its own packet filtering and forwarding-table lookup capabilities, the physical router’s recollection desires to be distributed or be allocated among the virtual routers. For the trie-based algorithms, the number of trie nodes can be taken as assess of the memory required, given a repaired node dimensions that normally consumes one or more recollection phrases. If packet-forwarding facts and figures

organisations are retained separately, the memory dimensions can become prohibitive stopping important scaling of the virtual router capability. As a plaything demonstration, if we have two virtual routers with the aforementioned prefix benches, we need the memory to be adept to hold 12 trie nodes since each trie has 6 nodes.

This design apparently does not scale well. To find a more scalable answer, we address the following difficulty: how to shop multiple binary endeavors in a compact blended facts and figures structure.

TRIEBRADING

We evolve trie braiding as a means to complete a more compact blending of the one-by-one endeavours, even when the one-by-one tries are not primarily similar. By using a single braiding bit in a trie node, we can turn around the meaning of its child pointers: When the bit is set, the left pointer represents the "1" agency and the right pointer comprises the "0" agency. We call this mechanism braiding because we can without coercion swap the left and right subtries of any trie nodes.

Trie braiding permits us to adjust the form of dissimilar endeavors and make them as alike as likely so that the number of distributed nodes is maximized (thereby reducing memory desires) upon merging. To show, in **Figure 1b**, if we swap node a's child nodes and node c's progeny nodes, the resulting trie will become alike to the trie in **Figure 1a**. Now we can amalgamate these two endeavors into one and get a new trie as shown in **Figure 2.** This trie has only 7 nodes. In supplement, for this case, leaf impelling will not increase the total number of nodes.

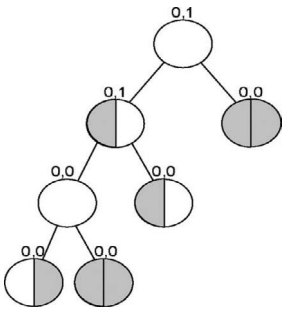

Figure 2. Node sharing using trie braiding. The numbers are the braiding bit value for the original tries.

One time the value of the braiding bits is very resolute, to insert a prefix in the trie (or to lookup a given IP address), we do the following. Beginning from the origin node, we analyze each prefix bit sequentially. We contrast this bit to the braiding bit at the current trie node. If they are equal, we pursue the left pointer; else, we pursue the right pointer.

The braiding morsels are exclusive to each virtual router. When we amalgamate tries from multiple virtual routers, the braiding method donates us a helpful device for minimizing recollection usage by maximizing the node distributing **(Table 2).**

Table 2. Notations.

T_i	Tree $_i$
v_i	Generic node in tree T_i
h_i	Depth of tree T_i
r_i	Root of tree T_i
$P(v_i)$	Parent of node v_i
$C_L(v_i)$	Left child of node v_i
$C_R(v_i)$	Right child of node v_i
$d(v_i)$	Depth of v_i
$t(v_i)$	Subtree rooted at v_i
$n(v_i)$	Number of nodes in $t(v_i)$, or Weight of v_i
$N_i(h)$	Set of nodes at depth h in T_i
$U_i(h)$	Number of isomorphisms at depth h *in* T_i
$L(v_i)$	Isomorphism label of node v_i
$\|T_i\|$	Size of Tree I, i.e. $n(r_i)$
$\Delta(v_i, v_j)$	Distance between $t(v_i)$ and $t(v_j)$
$S(v_i, v_j)$	Optimal mapping track bit
B_i	Braiding bit at node v_i

OPTIMAL TRIE BRAIDING ALGORITHM

Braid : A Dynamic Programming Algorithm

We now give a dynamic programming-based mapping algorithm (BRAID) that determines the optimal braiding pattern to minimize the allowance of memory to store the two trees. The input to the optimal braiding algorithm is the two binary trees, and the output from the algorithm is the following

I. a minimum-cost mapping M fromT2 to T1;

II. a braiding bit B(v2) for each nonleaf node v2€T2 that indicates the straight mapping (B(v2)=0) or the twisted mapping

(B(v2)=1) applied to node v2 's two child nodes.

Computing Mapping Charges

Given tree Ti rooted at ri, our aim is to work out Δ(r1,r2) . This can be accomplished from the base up.Since any node in the tree has at most two children, we can compute the worth of Δ(v1,v2) by contemplating the following two mapping possibilities:

- a straight mapping :Δ(v2, v1) where the left (right) progeny of v2 is mapped to the left (right) child of v1
- a twisted mapping: Δ(v2,v1) where the left (right) progeny of v2is mapped to the right (left) child of v1.

First, we use a depth-first search on each tree separately and compute the depth of all the nodes in both trees. Let hm=min {h1, h2}, Nodes with depth larger than hm will be mapped to We let the nonexistent node be , for which n(v)=0. We can now define the expanse between two nodes v1€T1 and v2€T2, Δ(v1,v2)=|n(v2)-n(v1)|

Determining the Minimum-Cost Mapping

Now the algorithm begins to determine the minimum-cost mapping for each node in T2. It begins from r2 and uses S (,) to get the optimal mapping. In supplement, the algorithm outputs the braiding bit B (v2) for each node. If B (v2) =0, then the straight map is optimal at node. If B (v1) =0, then the twisted map is optimal at node.

Fast Braid: Braiding With Isomorphism Detection

We advance the presentation of BRAID considerably both in periods of running time as well as memory utilisation by identifying the isomorphic subtrees during the course of running the algorithm.The performance improvement with this modification is determined by the following:

- Time taken to identify isomorphisms;
- The number of isomorphic subtrees at each depth.

Generating Labels

Two nodes will be given the same mark if and only if the subtrees fixed at those two nodes are isomorphic.

Determining the Distance

In the BRAID algorithm, we calculated Δ(v2,v1). In the FAST-BRAID algorithm, we have to calculate the value of Δ for two labels, each from a different tree.

K- Braid : A-Step Lookaheadheurestic For Braiding

In situations where working out the optimal braiding answer is time-consuming, we can use a -step lookahead heuristic to compute the mapping. Unlike the optimal braiding algorithms that compute the mapping utilising a bottom-up approach, this heuristic algorithm works out the mapping from origin to leaves.The heuristic is founded on a very easy idea: If we do not have information of the forms of the subtrees, the best scheme is to chart heavier subtree to heavier subtree and lighter subtree to lighter subtree. The parameter is the lookahead steps that we use to improve the accuracy of our estimation.

At all step, the lookahead design imaginarily truncates the tree to a deepness of from the current origin .We then run BRAID on these “truncated” trees to work out the optimal strategy at the origins (i.e., if the two subtrees of one origin should be braided or not). By doing this, the whole tree structure underneath deepness is abstracted by a lone weight value. After each step, the mapping of the progeny nodes of the origins is repaired. Then, at each of the child nodes, we replicate the method until all the tree nodes are mapped.

In generic step while considering nodes v1€T1 and v2€T2:

Truncate trees T1 and T2 at depth at d(v2)+k; Determine the leaf node weights

n(vi);

Run BRAID on the truncated trees;

Write braiding bit B(v2) to v2 and fix M(CL(v2)) and M(CR(v2)) .

This procedure is repeated for all nonleaf nodes v2€T2.

Combining Multiple Trees

So far, we have administered with the difficulty of combining two trees. If we want to blend more than two trees, then the running time of the optimal algorithm augments exponentially with the number of trees. For demonstration, if there are three trees T1, T2, T3, and we desire to chart T2andT3 onto T1. At depth h, we have to address the cost of mapping every pair of nodes (v2, v3), where v2 is at deepness of tree T2 and v3 is at deepness of tree T3, with every node v1 at deepness of tree T1. This makes the optimum algorithm prohibitively costly to run. Therefore, we use an incremental approach where we first amalgamate T1 and

T2 and then merge up T3 on this blended tree. Though it is not optimal, the running time only goes up linearly in the number of trees. The alternative of the alignment in which the trees are merged can make a distinction to the answer, but from our checking, we find that the distinction in the amalgamating order is negligibly little for all our tests.

CONCLUSION

Effective asset distributing while maintaining the logical isolation is an important design topic for virtual routers. In this paper, we present the trie braiding design and the optimal braiding algorithms to maximize the memory distributing for trie-based packet forwarding and filtering data organisations. The experiments on a real facts and figures set display our algorithms are highly effective on recollection decrease without compromising the presentation. Trie braiding is applicable when the data structure involves multiple trees, no matter for one router or for multiple virtual routers.

REFERENCES

1. Feamster N, et al. How to lease the Internet in your spare time, Comput. Commun.Rev. 2007; 37:61–64.
2. Global Environment for Network Innovations, GENI, 2006.
3. Turner J. A proposed architecture for the GENI backbone platform, in Proc. ACM/IEEE ANCS 2006;1 –10.
4. Bavier N, et al. In VINI veritas: Realistic and controlled network experimentation, in Proc. ACM SIGCOMM 2006; 3–14.
5. Curran J. IPv4 depletion and migration to IPv6, 2008.
6. Nordmark E and Gilligan R. RFC 4213: Basic Transition Mechanisms for IPv6 Hosts and Routers 2005.
7. Intelligent logical router service, Juniper Networks, Sunnyvale, CA 2004.
8. Route Views project, Univ. Oregon, Eugene, OR, 2011.
9. Fu J and Rexford J. Efficient IP address lookup with a shared forwarding table for multiple virtual routers, Proc. ACM CoNEXT 2008.