A Novel Approach for Prefix Minimization using Ternary Trie (PMTT) for Packet Classification

Sanchita Saha Ray
Department of Information Technology
St. Thomas` College of Engineering & Technology
Khidderpore, Kolkata, India
e-mail: saharay.sanhcita@gmail.com

Abhishek Chatterjee
Department of Computer Science & Engineering
St. Thomas` College of Engineering & Technology
Khidderpore, Kolkata, India
e-mail: sweetabhishek2@gmail.com

Surajeet Ghosh
Department of Computer Science & Technology
Indian Institute of Engineering Science and Technology
Shibpur, Howrah, India
e-mail: surajeetghosh@ieee.org

***Abstract*—A novel approach for eliminating the redundant and overlapped prefixes from a prefix table is proposed here. This approach reduces the number of prefixes by merging two prefixes on satisfying some specified conditions and eliminating any one of them depending on the conditions satisfied by those two prefixes and also by eliminating duplicate prefixes at the time of tree creation. To make the system faster, a novel ternary-trie based minimization algorithm has been proposed in place of Espresso-II minimization technique which increases the entire system complexity super linearly with the increase in number of prefixes and also exponentially increases the required time to update the prefix table. The main objective of the proposed technique is to reduce the storage space requirement for a prefix table and thereby reduce power consumption and cost factor associated with TCAM based prefix table by a healthy margin. The proposed prefix minimization technique shows 62.5% reduction in routing table size.**

Keywords—Prefix Minimization, Packet forwarding, Prefix Matching, Packet classification, Ternary Trie

I. INTRODUCTION

In an attempt to control the eventual widespread connectivity breakdown due to exponential growth of global routing table size, ISPs assisted in keeping the table size as small as possible, by the deployment of Classless Inter-Domain Routing (CIDR) and route aggregation [1] during late 2001. This surely slowed the growth to a linear process for several years, but, with the expanded demand of multi-homing by the user networks, the growth is once again super-linearly increased to nearly 500,000 prefixes as in [2]. The deployment of CIDR, necessitates storing of any arbitrary length (largest prefix in IPv4: 32 bits and IPv6: 128) prefix in the IP forwarding table, which poses the main challenge in internet protocol (IP) address lookup for packet forwarding. As a result the identification of the best suited next hop (outgoing port) address for each incoming packet becomes difficult due to multi-match prefix problem in the IP lookup table (LUT). The matching technique adopted for selecting the best suited prefix in the routing table is called Longest Prefix Matching (LPM) [3]. It is seen that the number of possible routes in a routing table is typically much smaller than the total number of prefixes present in the routing table. It is observed for some backbone routers at major US Internet exchange point (IXP) that, at least five hundred times more prefixes are present compared to the existent routes in the routing table [1]. This imbalanced figure of existing routes and the number of prefixes present in the routing table demands for a steering approach to reduce the size of the routing table by removing the redundant prefixes from the table. Recently, several approaches have been proposed to forward packets using both trie-based and TCAM-based approaches.

The most adopted hardware solution to perform IP address lookup in high performance systems is the use of Ternary Content Addressable Memory (TCAM) based devices [1], [3], [4], [5], [6], [7], [8], and [9]. The TCAM based approaches can provide the excellent performance in terms of table look up capability, but suffer from some inherent shortcomings viz., less capacity, high price and very high power consumption. Thus, it is desirable to compact routing table size so that a smaller number of TCAM chips would be used in the system. As the number of routing prefixes is increasing steadily, therefore, an efficient prefix minimization algorithm is needed to reduce the required number of TCAM chips to store the prefixes and thereby reduce power. Researchers have proposed a few approaches to reduce power consumption in TCAMs, including routing-table compaction and some other techniques to eliminate the redundant prefixes by using Espresso-II minimization technique. However, this takes excessive time for update because the Espresso-II minimization algorithm increases the complexity of the entire system with the increase in number of prefixes in a routing table [10]. These logic minimization problems suffer from NP-complete problem and therefore, the solution provided by Espresso-II is a near optimal solution with finite computing resources [11]. This inspires to device an approach to provide exact solution which is presented in this paper.

978-1-4799-4075-2/14/$31.00 ©2014 IEEE

The main contribution of this paper is divided into two parts viz., firstly a novel ternary trie based data structure and secondly an efficient approach based on ternary trie to reduce the routing table size by removing the redundant and overlapping prefixes from the routing table. This paper is organized as follows. Section II briefs the existing prefix minimization techniques. Section III introduces the concept of a novel ternary trie. Ternary trie based prefix minimization technique is explained in Section IV. The performance of the proposed technique is evaluated in Section V and finally, Section VI presents some concluding remarks.

II. EXISTING PREFIX MINIMIZATION TECHNIQUES

There are few existing techniques in order to eliminate the redundant prefixes from the prefix table and many of them [8], [9] used Espresso-II minimization technique. But Espresso-II is a complex minimization technique where the minimization time complexity increases super-linearly with the increased number of prefixes which makes the system slower. In [10], the table entries are reduced using Espresso-II algorithm and the number of inputs to the Espresso-II algorithm is reduced using prefix overlapping technique as described in [11] and thereafter, three various techniques are used in parallel to further reduce the table size. In [11], another technique called prefix aggregation is introduced in order to further compact the routing table size. [1] suggested two different techniques to reduce the number of prefixes, the first one is 'Pruning', which eliminates redundant prefixes by identifying the parent prefix and another is 'Mask extension' which exploits the flexibility given by TCAM hardware based on Espresso-II. There are numerous other minimizations techniques [7], [12] and [4] available which are incapable of producing non-prefix ternary rules and thereby overlook the chance of further reduction and in [6] the concept of bit weaving was introduced.

III. A NOVEL TERNARY TRIE

Most of the prefix minimization algorithms are based on ESPRESSO-II minimization technique which is a complex minimization technique, where the minimization time complexity increases super-linearly with an increased number of prefixes which makes the system slower. Since in CIDR format, an IP address is represented using a prefix, therefore, we require ternary ('0', '1', 'X') information to represent them. In reality, no such trie exists which can represent ternary information. In this connection we proposed a novel trie which can represent ternary information called ternary trie, and also proposed an algorithm based on ternary trie to perform prefix minimization efficiently.

A binary trie is a bit-wise data structure, where each single bit value ('0' or '1') of a prefix is represented using an edge of the trie. In our proposed Ternary Trie (TT) structure, we have kept room for representing ternary information by introducing three different nodes under the same parent node in the tree. These three children of a node in the TT are viz., the left child specified by a bit value '0', the right child, by '1' and the middle child indicated by 'X' (don't care) as shown in Fig.1. So, the degree of each internal node in TT could be at most three. A ternary trie is an ordered tree data structure that is used to store an associative array where the keys are ternary strings. The internal nodes in the trie by no means store the keys (prefixes) which are associated with those nodes, and instead, their positions in the trie reveal all the keys which are associated with them. Therefore, an offspring of an internal node in the trie, share a common forepart of the prefixes associated with that node.

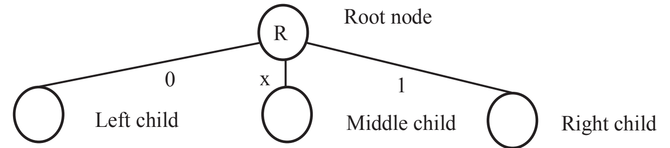

Fig. 1. A basic Ternary Trie (TT)

The trie creation procedure of TT is much similar to that of the binary trie, but the specialty in the ternary trie is the inclusion of the third node (middle node) as shown in Fig.2.

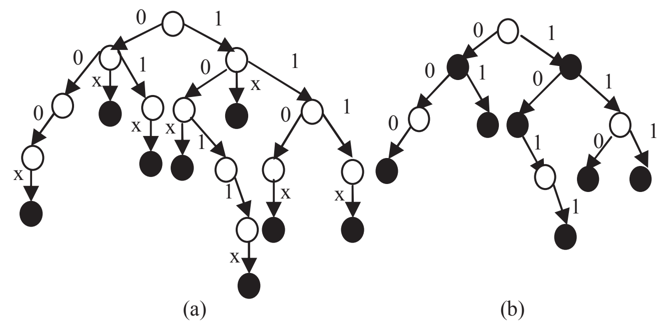

Fig. 2. (a) a ternary trie, (b) a binary trie, ● represents a prefix.

The tree structure of a ternary trie and a binary trie are shown respectively in Fig. 2(a) and 2(b), for the same set of eight prefixes viz., 000*, 0*,01*,10*, 1011*, 1*,110* and 111*. This could be observed from the above figures that, for a binary trie, a prefix could appear even at the internal nodes which introduces the difficulty in isolating the prefixes during tree traversal. Even in case of multi bit trie, the same prefix isolation problem prevails, therefore, in this paper, we proposed a novel ternary trie data structure to denote prefixes only by the leaf nodes of the tree.

IV. TERNARY TRIE-BASED PREFIX MINIMIZATION

As TCAMs are widely used memory chips for packet classification in networking, it is observed from above discussions that to reduce the power consumption and power dissipation of TCAMs, the number of TCAM entries must be reduced.

The proposed prefix minimization technique for packet classification uses the ternary trie-based approach to reduced the number of TCAM entries. In order to understand how the proposed ternary trie-based prefix minimization algorithm works and how the prefixes are minimized, we will explain using a sample prefix table with prefixes from three different hop-ids viz., 0, 1 and 2, shown in Table I. The prefixes in Table I are classified into three groups based on their hop-ids

namely G_0 (hop-id: 0), G_1 (hop-id: 1) and G_2 (hop-id: 2). Since, the hop-ids associated with the prefixes must be maintained even after minimization, therefore, we have chosen hop-id for the purpose of prefix-classification.

At first, the proposed algorithm groups all the prefixes according to their next hop id and then each group is treated separately for the creation of the ternary tree for that group in order to reduce the number of prefixes. Now, in order to explain the operational functionality of the novel ternary trie-based prefix minimization technique, let's choose prefixes from G_0 prefix group which contains ten prefixes viz., 01001x, 0101x, 01101x, 0111x, 1000x, 1001x, 11101x, 111100x, 111110x and 111111x. The length of the longest prefix in this prefix group determines the depth (no. of levels) of the tree. In this particular example, the size of the longest prefix is seven and hence, the ternary-tree formed using prefixes of G_0 group has seven (excluding root, which is considered at L0) different levels from L1 through L7. In the proposed trie structure, all the prefixes are made equal length by appending 'X' at their end. Similarly, separate trees are formed using G_1 and G_2 group of prefixes those would have 4 and 6 different levels respectively. The working of tree minimization algorithm is done in two phase viz., tree creation phase and tree merging phase. However, in the first subsection of this section, the properties of the ternary trie has been described which is used to build the prefix minimization tree.

TABLE I. SAMPLE PREFIX TABLE AND GROUPING OF PREFIXES BASED ON HOP-IDS

Before Grouping		After Grouping		
Prefixes	**Next Hop ID**	**Prefixes**	**Next Hop ID**	
01001x	0	01001x	0	G_0
0101x	0	0101x	0	G_0
01101x	0	01101x	0	G_0
01101x	1	0111x	0	G_0
01111x	1	1000x	0	G_0
0101001x	2	1001x	0	G_0
10011x	2	11101x	0	G_0
11101x	2	111100x	0	G_0
0111x	0	111110x	0	G_0
1000x	0	111111x	0	G_0
1001x	0	01101x	1	G_1
11101x	0	01111x	1	G_1
111111x	2	10000x	1	G_1
10000x	1	0101001x	2	G_2
111100x	0	10011x	2	G_2
111110x	0	11101x	2	G_2
111111x	0	111111x	2	G_2

A. *Ternary trie properties*

The proposed algorithm uses a ternary trie to eliminate the overlapping prefixes and merges two prefixes if they differ by 1-bit at any level. The novel ternary tree data structure has the following properties-

i) Each node of the tree contains five parts namely level, data, left child, right child and middle child. The level contains information regarding the position of the bit in the data structure, counting of level starts from 0 (root) and keeps on increasing as the tree grows; data part of a node in level i contains the value of $(L-(i-1))^{th}$ bit (where, L is the length of the longest prefix) of prefixes associated with that node; left child, right child and middle child contains pointer to the node's left, right and middle child respectively.

ii) The root node of the tree contains a dummy value say 'R'.

iii) Each node of this tree can have at-most three children namely left child, right child and middle child. The left child of a node contains a value '0', right child contains a value '1' and the middle child contains a value 'X' (X: don't care).

iv) Each unique path from level 1 (taking root as level 0) through leaf node of the ternary tree represents an unique prefix for a certain prefix group.

B. *Ternary Tree Creation Phase*

The prefixes of a certain group are stored in a 2-D array '$G_K[i][j]$' where K is prefix group number and 'i' denotes the prefix number with $0 \le i < N$ (N: number of prefixes) and 'j' denotes the bit position in a certain prefix with $0 \le j \le L$ (L: length of the longest prefix in the considered prefix group) in a certain prefix group.

The tree creation procedure begins with root node formation and a dummy value (say 'R') is assigned to the root node. The novel ternary tree creation process needs a pointer 'Ptr' to traverse the tree in order to find the proper position for inserting a node in a certain level of the tree. At first, the pointer 'Ptr' is initialized with the address of the root node. If the MSB of a certain prefix is found to be 0, then the prefix would be inserted in the left sub-tree and if it is found to be 1, then it would be in the right sub-tree and else (MSB: X) it would be in the middle sub-tree of the root. This process continues for all the bits of a prefix. The recursive procedure for the ternary tree creation is given below.

Ternary_Tree_Creation Algorithm:

```
createNode(newValue, TreePtr &Ptr)
begin
    TreePtr newNode;
    if (Ptr == NULL)
        newNode = new TreeNode;
        newNode -> name = newValue
        newNode -> left = NULL;
        newNode -> middle = NULL;
        newNode -> right = NULL;
        Ptr = newNode;
    else if (newValue == 0)
        createNode(newValue, Ptr -> left);
        else if (newValue == X)
            createNode(newValue, Ptr -> middle);
        else
            createNode(newValue, Ptr -> right);
end
```

After inserting a single prefix in the tree, the value of the three pointers 'Ptr' are initialized to the address of the root node and the same process as described above is repeated for all other prefixes in a particular prefix group (G_k). Now, if two prefixes are identical then only the first one amongst them is

inserted and this automatically removes the duplicate prefix a at the time of tree creation with no extra effort for the detection and removal of those prefixes. In Fig. 3 a ternary-trie has been created with prefixes from G_0 prefix-group of TABLE I.

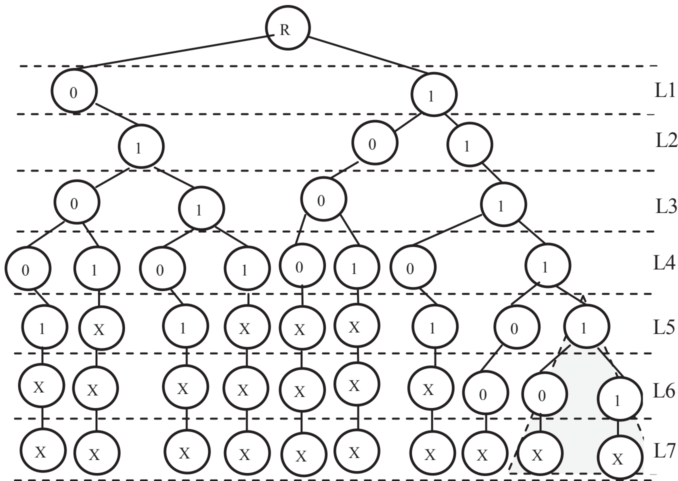

Fig. 3. Ternary-Trie using prefixes of G_0 Group

C. *Ternary -Tree Merging Phase*

Once the tree has been created, thereafter the tree minimization process starts. The minimization process begins at (leaf-1)th level and continues until (root-1)th level. Any two nodes from the same level of the tree could be merged if those satisfy Condition I and any one of the following four conditions as stated in Condition II.

Condition I. The two nodes which could be merged together must belong to the same parent node.

Condition II. Nodes those could be merged together, must satisfy any one of the following four conditions.

i) If those two nodes have one identical sub-tree (left, right or middle sub tree).

ii) If those two nodes have identical combination of any two sub-trees of their own.

iii) If those two nodes have identical left, right and middle sub-trees.

iv) If those two nodes are the leaf nodes of the tree.

After establishing the criterion for merging of any two nodes at a given level of the constructed tree, now the actual procedure of prefix minimization would be explained. The original prefix tree needs to be reformed in order to reflect the changes after satisfying the merging criterion for any two nodes for the reduction (merging) of the tree.

Trie Merging Procedure:

Step 1: The rules for transforming the tree are given below.

a) If any two nodes satisfy criterion I and either of II (i) or II (ii), then again sub-tree formation can have few cases depending upon the presence of the middle node.

Case I: If the parent node of those two nodes doesn't have a middle child, then a new node (middle node) under the same parent is created. This newly created middle node inherits only the identical sub-tree part(s) of those two nodes. However, their dissimilar sub-tree parts are kept intact with their respective nodes under the same parent node.

Case II: If none of the two nodes which could be merged, is a middle node and the parent node of those two nodes has a middle child, then these two node's identical sub-tree(s) are removed by inserting them under the pre-existent middle node of their parent node. However, their dissimilar sub-tree part(s) are kept intact with their respective nodes under the same parent node. However, insertion of the new sub-tree(s) under the middle node should not conflict with the existing sub-trees of that node and no duplicate nodes are inserted at any level of the sub-tree.

Case III: If one of the two nodes (which are to be merged together) is a middle node, then only the identical sub-tree part(s) is removed from the node other than the middle node, however, the unmatched sub-tree part(s) are kept intact with that node.

b) If those two nodes satisfy criterion I and II (iii), then again sub-tree formation can have few cases depending upon the presence of the middle node.

Case I: If the parent node of the two (left and right) nodes which are to be merged together, doesn't have a middle child, then directly those two child nodes are removed by merging them into a single node and designated as the middle node of their parent node. The newly created middle node would contain the sub-trees of any one of those two removed nodes.

Case II: If none of the two nodes which could be merged is a middle node, and the parent of those two nodes have a middle child, then these two nodes along with their sub-trees are directly removed by inserting their sub-trees under the pre-existent middle node of their parent. However, the insertion of the new sub-tree(s) in the middle node no way conflict or duplicate the pre-existent sub-trees of that node.

Case III: If one of the two nodes (which are to be merged together) is a middle node, then the node other than the middle node is removed completely from the tree.

c) If those two nodes satisfy criterion I and II (iv), then the procedure of sub-tree formation after merging of nodes at leaf level is given below. Here, the only node which could appear as a leaf node for any parent node in the tree is the middle node. This

violates the condition stated in (I) and therefore, no merging of nodes at this level is possible. sibling

Step 2: Step 6 is repeated from the leaf level to the level before the root level.

Step 3: Step 4 through Step 6 is repeated for all the prefix groups.

Step 4: All the resultant groups are merged to get the minimized prefix table.

To illustrate the idea of prefix minimization using ternary trie-based approach, let us consider the prefixes from Fig.3. The merging procedure starts checking from (leaf-1)th level i.e., from 6th level, as it is already seen that, minimization at leaf level is not possible.

Minimization at Level 6:

At level 6 of the original ternary-tree (shown in Fig.3 by shaded triangle), it is observed that the 9th and the 10th nodes at L6 from L.H.S to R.H.S having identical middle sub-tree and moreover, these two nodes belong to the same parent node at L5. Therefore, these two nodes could be merged together into a single node and their identical sub-tree part(s) comes under this newly created node, which is indicated by the middle node ('X') of their parent node. However, as those two nodes at Level 6 don't have any other dissimilar sub-tree(s), so these nodes are permanently removed from the tree. No further minimizations could be done at this level and the resultant tree after minimization is shown in Fig. 4. If the tree is traversed from level 1 through level 7, then, all the unique prefixes we have are 01001x, 0101x, 01101x, 0111x, 1000x, 1001x, 11101xx, 111100x and 11111xx.

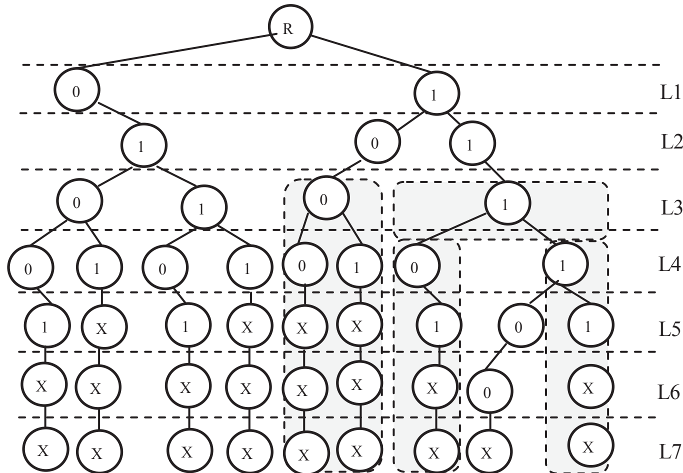

Fig. 4. Tree after minimization at L6

Minimization at Level 5:

Now, the algorithm will move backward to level 5 and at this level no two nodes belonging to the same parent node having identical sub-tree(s) as we can see from Fig. 4. Therefore, minimization at this level is not possible, so, it would provide same result as we receive at level 6.

Minimization at Level 4:

At this level, we find two pairs of nodes for merging which is shown using shaded portion in Fig. 4. The first pair of nodes from L.H.S to R.H.S is the 5th and 6th nodes in L4, those having identical middle sub-tree and therefore, could be merged to a single node with the identical middle sub-tree under it. Similarly the second pair of nodes is 7th and 8th nodes from the L.H.S to R.H.S in L4 having identical right sub-tree and therefore, merged into a new node, indicated as middle node of their parent with their common right sub-tree part under this newly created node. However, as it can be seen from Fig.4, that the 8th node (at L4) also having a left sub-tree, therefore, this node is kept along with its left sub-tree under its parent node at L3 shown in Fig. 5.

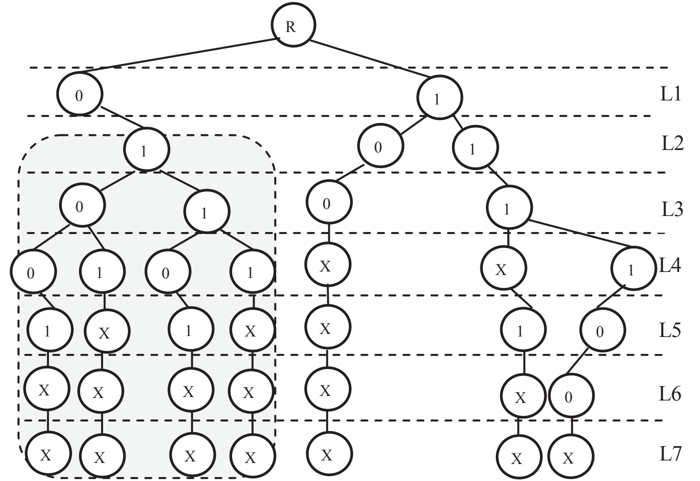

Fig. 5. Tree after minimization at L4

Minimization at Level 3:

At this level, we find one pair of nodes for merging which is indicated by the shaded portion in Fig. 5. The pair of nodes which could be merged in L3 is the 1st and the 2nd node from L.H.S to R.H.S. These two nodes of L3 having identical left and right sub-tree and therefore, could be merged to a single node indicated by the middle child of their parent node with the same left & right sub-trees under it. Since, these two nodes of L3 don't have any other mismatched sub-tree part, hence, these nodes are entirely replaced from the original tree by the inclusion of the newly created node (middle node indicated by 'X') under their parent node, which is shown in Fig.6.

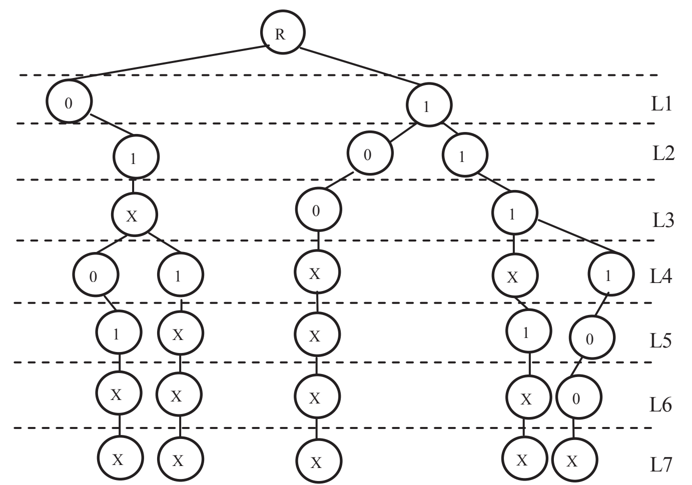

Fig. 6. Tree after minimization at L3

Minimization at Level 2 & 1:

Now, the algorithm will move backward to level 2 and it is clear from Fig.7 that, at this level no two nodes belonging to the same parent node with identical sub-tree(s) are found, as a result, minimization at this level is not possible. Therefore, no further minimization in the number of prefixes after the minimization at L3 is made, and so after minimization at this level (L2) it would provide same prefixes as we received at level 3. Similarly, at level 1 also, no two nodes belonging to the same parent node (root node) having identical sub-tree(s) and consequently, minimization in terms of prefix merging at this level also not possible.

TABLE II. MINIMIZED NUMBER OF PREFIXES AFTER APPLYING PREFIX MINMIZATION TECHNIQUE ON G_0 GROUP OF TABLE I

Prefixes							
Before Minimization	After Minimization at various levels						
	L7	L6	L5	L4	L3	L2	L1
01001x		01001xx		01001xx	01x01xx		
0101x		0101xxx		0101xxx	01x1xxx		
01101x		01101xx		01101xx	100xxxx		
0111x		0111xxx		0111xxx	111x1xx		
1000x	--	1000xxx	--	100xxxx	111100x	--	--
1001x		1001xxx		111x1xx	-		
11101x		11101xx		111100x	-		
111100x		111100x		-	-		
111110x		11111xx		-	-		
111111x		-		-	-		

As a result, the tree after minimization at L3 is the final minimized tree and the minimized set of prefixes is shown in Table II. Thus, the final set of prefixes in G_0 group are 01x01xx, 01x1xxx, 100xxxx, 111x1xx and 111100x, which shows 50% reduction in the number of prefixes compared to the total number of prefixes present before minimization.

V. SIMULATION RESULT

The proposed ternary-trie based prefix minimization algorithm is tested for some publicly available Border Gateway Protocol (BGP) [2] routing tables from the public networks as MAE-EAST, MAE-WEST, AADS and PacBell. These different test cases with different capacity are listed in Table III. It can be noticed from the following Table that, the percentage (%) of prefix minimization shows up to 62.5, which saves TCAM space, number of active TCAM cells, power consumption with a healthy margin.

TABLE III. SIMULATION RESULTS FOR VARIOUS TEST CASES

Test Case	No of Prefixes		% Minimization $P=((N-N_1)/N) * 100$
	Before Minimization (N)	After Minimization (N_1)	
Mae-East	353301	162520	54
Mae-West	442090	214856	51.4
Pacbell	238519	99462	58.3
Aads	437933	164224	62.5

VI. CONCLUSION

It is evident from Table III that the prefix reduction or minimization factor primarily depends on the nature of the prefixes than the number of prefixes present for minimization. Though, there could be the possibility of more number of redundant prefixes to appear as the number of prefixes increase in the prefix table for a fixed prefix length. The ternary trie based prefix minimization approach proposed here shows up to 62.5% prefix table size reduction. This routing table size reduction directly saves a large amount of storage space and thereby shows the reduction in power consumption and cost factor of the TCAM based prefix table by a healthy margin. Thus, the TCAM storage space saved due to the reduction in the number of prefixes could have following advantages: firstly the number of TCAM entries needed to store the prefixes are reduced, the cost is also reduced due to either lesser number of TCAM chips or a smaller capacity TCAM chip could be used. Secondly, the number of active cells at the time of comparison will also be reduced so, the power consumption is also reduced. Moreover, the space thus saved could be used for accommodating new prefixes in the rule table. Since, the proposed prefix minimization technique is trie based data structure so, only the tree traversal and some extra condition checking are needed for this technique, which makes this algorithm more efficient and results in high throughput compared to the techniques using Espresso-II minimization technique. Furthermore, this technique could be easily deployed in the existing system without significant changes and would be suitable for both IPv4 and IPv6 prefix format.

REFERENCES

[1] Huan Liu, “Reducing Routing Table Size Using Ternary-CAM”, Hot Interconnects 9, pp. 69-73, 2001.

[2] Potaroo - BGP Table data, http://bgp.potaroo.net/index-bgp.html, Accessed on 30th June 2014.

[3] S. S. Ray, A. Chatterjee, S. Ghosh, "A Hierarchical High-throughput and Low Power Architecture for Longest Prefix Matching for Packet Forwarding", Proc. of IEEE International Conference on Computational Intelligence and Computing Research, pp. 628-631, 2013.

[4] C. R. Meiners, A. X. Liu, and E. Torng. “TCAM Razor: A systematic approach towards minimizing packet classifiers in TCAMs”, In Proc. IEEE ICNP, pages 266–275, October 2007.

[5] A. X. Liu and M. G. Gouda, “Complete redundancy removal for packet classifiers in TCAMs”, IEEE Transactions on Parallel and Distributed Systems Vol. 21, No. 4, pp. 424-437, April 2010.

[6] Chad R. Meiners, Alex X. Liu, Eric Torng, “Bit Weaving: A Non-prefix Approach to Compressing Packet Classifiers in TCAMs”, In Proc. of the 17th IEEE ICNP, 2009.

[7] Q. Dong, S. Banerjee, J. Wang, D. Agrawal, and A. Shukla, “Packetclassifiers in ternary CAMs can be smaller”, In Proc. ACM Sigmetrics, pages 311–322, 2006.

[8] A. Mahini, G. Branch, R. Berangi, S. Fatemeh, Khatami Firouzabadi “Low Power Tcam Forwarding Engine for IP Packets”, IEEE Military Communications Conference, MILCOM, pp. 1 - 7, 2007.

[9] V. C. Ravikumar, Rabi N. Mahapatra, “TCAM Architecture forIP lookup Using Prefix Properties,” IEEE Micro, vol. 24, no. 2, pp. 60-69, April 2004.

[10] R. K. Brayton, G. D. Hachtel, C. T. McMullen, and A. L. Sangiovanni-Vincentelli, “Logic Minimization Algorithms for VLSI Synthesis. Kluwer”, Academic Publishers, 1984.

[11] R. Rudell and A. Sangiovanni-Vincentelli, “Espresso-MV: Algorithms for multiple-valued logic minimization”, In Proc. of the IEEE 1985 Custom Integrated Circuits Conference, pp. 230-234, 1985.

[12] R. Draves, C. King, S. Venkatachary, and B. Zill. “Constructing optimal IP routing tables,” In Proc. IEEE INFOCOM, pages 88–97, 1999.