MDTC: An Efficient Approach to TCAM-based Multidimensional Table Compression

Hanqing Zhu*, Mingwei Xu*, Qing Li†, Jun Li‡, Yuan Yang*, Suogang Li§,
* Tsinghua National Laboratory for Information Science and Technology (TNList)
Department of Computer Science and Technology, Tsinghua University
† Graduate School at Shenzhen, Tsinghua University ‡ University of Oregon § CERNET National Network Center
Email: zhuhanqing2011@gmail.com, xumw@tsinghua.edu.cn, yangy@csnet1.cs.tsinghua.edu.cn,
li.qing@sz.tsinghua.edu.cn, lijun@cs.uoregon.edu, lsg@cernet.edu.cn

***Abstract*—Ternary Content Addressable Memory(TCAM)-based multidimensional tables are widely used to implement Access Control Lists (ACLs) for Internet packet classification and filtering, and have also become attractive for constructing the forwarding tables of Internet routers and the flow tables of Openflow switches, where multiple fields are generally used to match incoming packets. However, as such tables can grow quickly as the Internet develops fast, and sometimes even expand in size because of TCAMs limitation in storing rules with range fields, it becomes imperative to compress these tables.**

In this paper, we propose a fast and efficient approach to multidimensional table compression. We divide the multidimensional space iteratively to obtain a series of cells, and then combine those cells that are associated with the same action. Our approach applies to tables of any dimension, addresses the range expansion problem, and provides efficient compression for TCAM-based tables. The experiments show that our approach has low computing cost in time, which is significant for the online update of tables. On average, it reduces 23.0% entries of the real-life ACLs, 25.8% to 55.1% of the generated two-dimension tables, 55.1% of the generated ACLs, and 28.4% of the generated Openflow flow tables.

I. Introduction

As the Internet has grown to be a worldwide complex distributed system, a notable phenomenon is that many key functionalities and services on the Internet depend on multidimensional tables. Such tables are not only seen in traditional functionalities such as packet classification, access control, and load balancing, but also in newly proposed networking architectures such as software-defined networking (SDN) [1] and two-dimension routing [2].

A multidimensional table is a sequence of predefined rules, where each rule consists of several *key* fields and one *action* field, with the corresponding action to be performed on the packets that match the key fields. For example, an access control list (ACL) [3] can be a five-dimension table where the key fields are the source IP prefix, the destination IP prefix, the source port, the destination port and the protocol type, and the action field indicates the access decision. A two-dimension routing table [2] can use source IP prefix and destination IP prefix as its key fields, with its action field to indicate the next hop for packet forwarding. The flow table of Openflow [4] even introduces more than ten dimensions. Clearly, different from a traditional routing table, which has only one key field of destination IP prefix, a multidimensional table can enforce more refined traffic control on the Internet.

A multidimensional table can be implemented using Ternary Content Addressable Memories (TCAMs). TCAMs encode prefix-specific rules with ternary state and support fast, parallel packet lookup in the table. For example, most ACLs are stored in TCAMs to achieve high-speed classification; Yang et al. designed two-dimension routing with TCAMs [2]; and Openflow entries are often installed in TCAMs as well [5].

However, multidimensional tables are faced with the problem of table explosion. As the number of Routing Information Base(RIB) entries in backbone routers has reached 500,000 [6] by July 2013, which is still growing, multidimensional tables based on such a routing table may have many more entries. For example, a two-dimension routing table would need the square of 500,000 entries in the worst case. On the other hand, TCAMs have limited storage, and the chips that support high-speed lookup are both expensive and highly energy-consuming. Besides, TCAMs suffer from the problem of range expansion problem. Because some fields of the multidimensional table are specified as range, the number of rules may increase severely when converting these rules to prefix specified rules. It therefore has become a critical problem to compress multidimensional tables for TCAMs. We focus on this problem in this paper. When two tables have the same action definition for each packet, they are equivalent. Specifically, given a multidimensional table, we study how to find an equivalent TCAM-compatible table with fewer entries.

There have been studies on compressing network tables with specific number of dimensions. The optimal routing table constructor (ORTC) [7] was proposed to compute the minimal table in the one-dimension case. Suri et al. [8] proposed an optimal compressing algorithm for special two-dimension tables, but the general problem for compressing two-dimension tables was proven to be NP-hard [9]. Many approaches have also been developed for five-dimension ACL tables [10–13].

The research is supported by the National Basic Research Program of China (973 Program) under Grant 2012CB315803, the National Natural Science Foundation of China under Grant 61133015, Specialized Research Fund for the Doctoral Program of Higher Education under Grant 20120002110060.

ISBN 978-3-901882-68-5

While all these algorithms are outstanding for their high compression ratio, each of them is subject to one or more of the following limitations: the algorithm is limited to specific dimension table compression; the computing overhead is high; and the range expansion problem still exits because we have to convert the output with range specified fields to TCAM-compatible format.

In this paper, we propose a fast and efficient approach called Multidimensional Table Compression(MDTC) to translate a big table into an equivalent and smaller one that can be stored in TCAMs directly. We use a box to represent an entry in the table and a multidimensional table is composed of a set of boxes. As the value of each field has a range, the table corresponds to a limited multidimensional space. MDTC divides the space iteratively and gets a set of small boxes, then combine the small boxes that have the same action.

Our approach has several significant benefits. Firstly, MDTC addresses the range expansion problem well. After converting rules to prefix-specific rules, the table expands. MDTC compresses the expanding table and produces a smaller one. Our experiments show that MDTC always produces a smaller one. On average, it reduces 23.0% entries of the real ACLs, 25.8% to 55.1% of generated two-dimension tables, 55.1% of generated ACLs, and 28.4% of generated Openflow flow tables. Second, it can be applied to any dimensional tables, and has a good compression efficacy on ACLs, flow tables, two-dimension routing tables. One can just use MDTC to compress all different tables in the same router. Third, MDTC compresses tables efficiently with a low computing cost. Compared to Diplomat [13], one of the most outstanding compression algorithms, the computing cost of MDTC is much lower.

The rest of the paper is organized as follows. Section II is an overview of MDTC. MDTC is presented in detail in Section III and IV. We evaluate the performance of MDTC with simulation in Section V. In Section VI, we further discuss the issues of lookup and updates. Section VII and VIII are the related work and conclusion, respectively.

II. Problem Statement of MDTC

A. Geometric Description of Table

A multidimensional table is essentially a list of entries with several columns, which is not so intuitive to do further work of compression. We thus interpret a multidimensional table using a multidimensional space where there are many boxes corresponding to the table entries, with each box defined by the fields of an entry and labeled with a corresponding action. If necessary, other information will be added to the box, like priority.

Specifically, a box of two-dimension table is a rectangle. Assuming in a coordinate system, horizontal ordinate and vertical ordinate denote the source address and destination address respectively, and $(*, *)$ means the whole space. A $(sourceaddress, destinationaddress)$ pair corresponds to a point and a (Ps, Pd) pair corresponds to a rectangle, where Ps and Pd are a source IP prefix and a destination IP prefix, respectively. For example, an IPv4 two-dimension routing table corresponds to a $2^{32} \times 2^{32}$ plane, and in this plane, $(1*, 0*)$ stands for a rectangle whose horizontal ordinate ranges from 128.0.0.0 to 255.255.255.255 and vertical ordinate ranges from 0.0.0.0 to 127.255.255.255 (see illustration in Fig.1). Absolutely, we can extend this geometric description to any dimension table.

Fig. 1. The gray rectangle is (1*, 0*). The action is labeled on it.

B. The Compression Problem

As each entry can be interpreted by a box, we can abstract the multidimensional table compression problem as follows: *given a list of boxes labeled with actions, find an equivalent list of boxes where the number of boxes is smallest.* To define the problem formally, denote D the multidimensional space corresponding to the table, denote L the list of boxes which are defined in the original table, and denote $|L|$ the number of boxes in L. Further denote $Match(P, L)$ the function that calculates the action that point *P* matches in *L*. For a L, we may find out another L' which satisfies: for each point $P \in D$, $Match(P, L') = Match(P, L)$. We say that L is equal to L'. We then can define the multidimensional table minimization problem as follows:

Given L, find out L_{min} among all of $L's$ which are all equal to L, so that $|L_{min}| \leq |L'|$.

C. Overview of MDTC

As the optimal solution to the multidimensional table minimization problem is NP-hard [9], in this paper, we aim at designing an algorithm that is appropriate and efficient for multidimensional table aggregation and compresses the original table into one as smaller as possible. First, it reduces the size of multidimensional table sufficiently. Second, it is fast to compress the table to make itself more practical. Third, the operation on each packet remains the same as original table according to the principle of equivalence: the router deals with packets in the same way before and after compression. Fourth, it is a general compression algorithm that can compress arbitrary dimensional table in network devices.

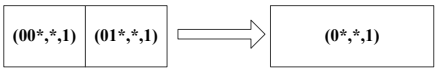

Fig. 2. Entry (00*, *, 1) and Entry (01*, *, 1) are left-right adjacent and can be merged into one entry (0*, *, 1).

The basic idea of compaction is using a bigger box to cover multiple smaller boxes with the same action. In multidimensional space, we label a bigger area with prevalent action,

and reduce multiple specific areas with the same action. For example, in two-dimension space, two adjacent areas that have the same action can be merged into one, as shown in Fig.2. Our algorithm contains two parts: dividing the multidimensional space into a series of no-overlap small boxes and combining the boxes that have the same action to achieve a small set of boxes.

III. MDTC Outline

A. Definition

We just consider the prefix-specific table in this paper, and MDTC is also available to range-specific table when we translate the range-specific fields into prefix-specific fields. Here are some concepts that will be used in the rest of the paper.

Definition 1. Related-box is the box corresponding to the entry in the original table, which will not be changed. And it can be presented as (prefix 1, prefix 2, ..., prefix n, action). When the original table includes priority which is not presented as prefix, the priority information will be added to the Related-box.

Definition 2. Space-box is used to describe the space covered by a box, and can be divided into a series of small boxes. And it can be presented as (prefix 1, prefix 2,..., prefix n), without $action$ compared to Related-box.

Definition 3. RBS is the shorthand of **Related-boxes set**, which is an attribute of a point. RBS is a set of Related-boxes which include the certain point. Generally speaking, one box also has RBS - the union of Related-boxes that are intersected with the specific box.

Definition 4. A **cell** is a box which meets two conditions as follows: (1) the points in the same cell have the same RBS; (2) each cell has no intersection with others. Because of the second condition, the points in the same cell have the same action that is decided by the match principle of different scenarios, like priority in packet classification.

Fig.3 is an example for these concepts. Rectangle (10*, *, 2) is a Related-box that corresponds to the fourth entry in the table. The square (*, *) is a Space-box, which is covered by all the Related-boxes. The RBS of (*, *) is {(0*, 1*, 1), (*, 01*, 1), (0*, 00*, 1), (10*, *, 2), (11*, 1*, 2), (11*, 00*, 2)}, which has a brief expression of {1, 2, 3, 4, 5, 6} when we give each entry (or Related-box) an index. In the third graph, rectangle (10*, 01*) is a cell with a RBS of {2, 4}. According to the destination-first Longest Prefix Match principle in two dimensional routing, we select the action of 1 for the cell. However, notice that the size of cells varies and what is important is that the points in the same cell must have the same action. As to packet classification, we select the proper action based on priority.

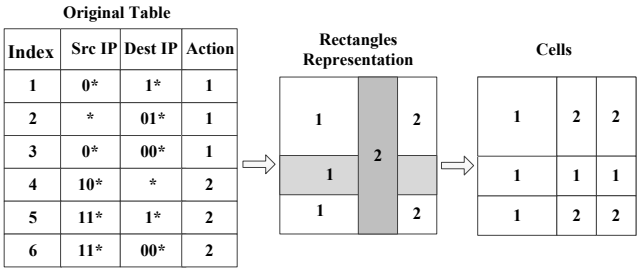

Fig. 3. A table is interpreted as rectangles and rectangle (*,01*) and (10*,*) overlap. The multidimensional space is divided into a series of cells.

B. A Compression Example

The input of our algorithm is a multidimensional table that is represented as a set of Related-boxes. The multidimensional space is defined according to the certain scenario. For example, the multidimensional space of a five-dimension ACL in Ipv4 with five fields is $[0, 2^{32}-1] \times [0, 2^{32}-1] \times [0, 2^{16}-1] \times [0, 2^{16}-1] \times [0, 2^{8}-1]$; these ranges correspond to source IP, destination IP, source port, destination port and protocol type respectively. And all the boxes are included in the space. As we look forward to reducing the entries, we want to replace the original set with another smaller set of boxes without changing the behavior of router. In short, MDTC contains two parts: converting multidimensional space into cells and then combining cells that have the same action.

We show a simple compression example on a two-dimension routing table in Fig.4. The upper are the input and the output. The nether is the process of MDTC algorithm. The gray part means that there are more than one rectangle covering the area. The first set of rectangles correspond to the original table directly, and the first rectangle is overlapped by the following three rectangles; the second set of rectangles are all cells after dividing; The third set is the result of combination with only two rectangles.

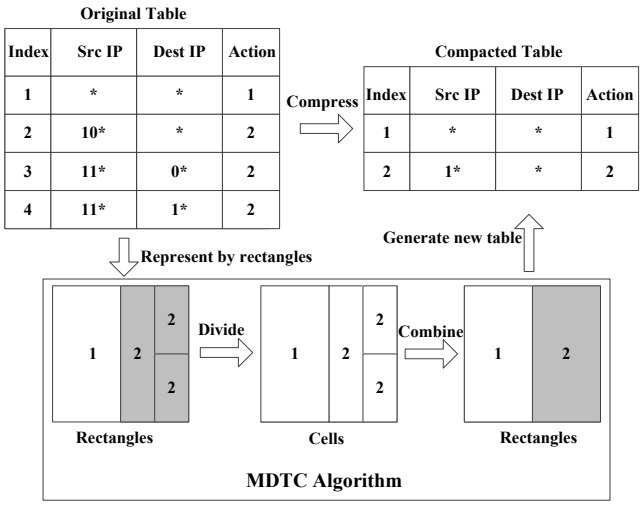

Fig. 4. Apply MDTC to a two-dimension routing table compression.

C. Dividing Multidimensional Space into Cells

Our algorithm starts with a multidimensional Space-box that contains all the Related-boxes in the original table and then divides it into a series of cells. These cells cover the space

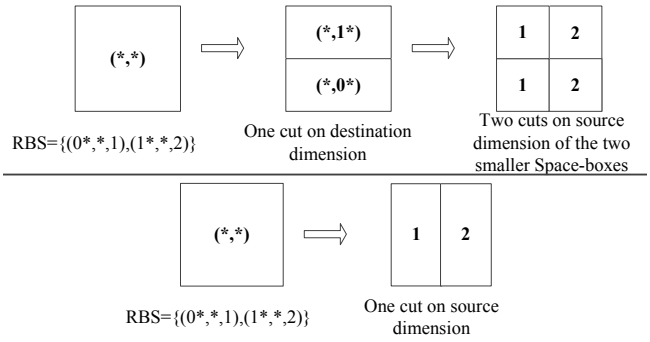

Fig. 5. Choose the dimension cutting by which has the smallest NOC. The upper chooses destination dimension firstly and needs two more cuts on source dimension. As a result, the cutting produces four cells. Meanwhile, the nether chooses the source dimension firstly and just divides the rectangle into two cells.

totally and keep the original action of each point. Given a Space-box, if it is a cell, we select a proper action from its RBS for it; otherwise, we choose a proper dimension to divide the Space-box evenly and obtain two Space-boxes to replace the divided one.

It is a challenge to choose a proper dimension to divide a Space-box into two equal-size smaller ones. When dividing, we take two principles into account: providing high opportunity for combining, because the combination reduces the entries finally; obtaining a set of cells as small as possible, because we merge the cells eventually and the computing overhead depends on the number of cells. According to the two principles, we propose two heuristic ideas to help the selection.

We observe that dividing a Space-box may cut its Related-boxes. The first idea is to take the number of cuts on the Related-boxes in RBS of the Space-box into account. We call the number as NOC, which is the acronym of number of cuts on the Related-boxes. When we divide the Space-box, we may cut several Related-boxes in the RBS, because the Related-boxes differ from each other in size and cover various part of the Space-box. And cutting by different dimensions may have different number of cuts. Intuitively, we prefer to cut in the dimension dividing by which results in the fewest cuts, for we are going to compact the cells at last and expect the number of cells to be smaller. Fig.5 is an example. Space-box (*, *) has two Related-boxes, (0*, *, 1) and (1*, *, 2). In this case, if we choose destination dimension then both of the two Related-boxes will be cut, and we need more cuts on the source dimension for both smaller Space-boxes to achieve cells; if we choose source dimension, neither Related-box will be cut, and we achieve two cells after one cut. As a result, the better choice generates only two cells while the other choice results in four cells.

Also, we observe each dimension may have different number of various prefixes, which has effect on the result. Here, we use NUP to denote the number of unique prefixes in the RBS of Space-box for certain dimension. The second idea is to take NUP into account when more than one dimension have the minimum cuts. If we do not divide the Space-box by the dimension with fewest unique elements in the earlier

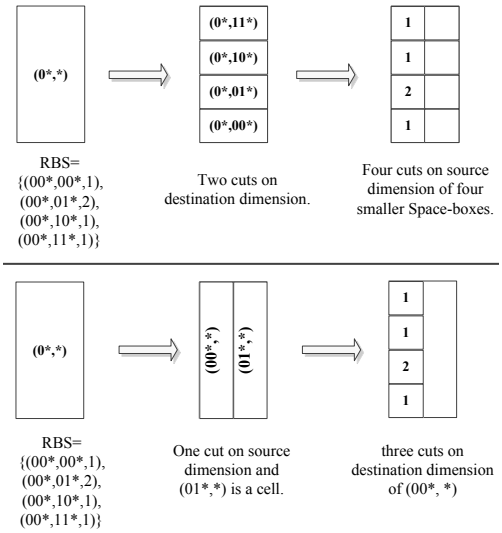

Fig. 6. The number of unique prefixes in source dimension is one while four in destination dimension. If we prefer to divide (0*, *) by destination dimension firstly, we get 8 cells finally, as shown in the upper; if we divide (0*, *) by source dimension firstly, then the number of cells decreases to 5, as shown in the nether.

period, we will divide many smaller Space-boxes by that dimension respectively and obtain a bigger set of cells. As shown in Fig.6, Space-box (0*, *) has four Related-boxes, and source dimension has only one unique prefix of {00*} while destination dimension has four unique prefixes of {00*, 01*, 10*, 11*}. If we firstly divide (0*, *) by source dimension, we get two small Space-boxes of (00*, *) and (01*, *). As (01*, *) is a cell, we do not need to divide it and we finally convert (0*, *) to 5 cells. If we divide (0*, *) by destination dimension firstly, we get two Space-boxes of (0*, 1*) and (0*, 0*), and neither of them is cell. Then, we have to divide both Space-boxes iteratively. As a result, we get 8 cells. So if we choose the dimension with smaller number of unique prefixes firstly, the set of cells will be smaller.

After a sequence division on the Space-box, we obtain a set of cells and these cells are labeled with action. Each cell may match several entries and the action selection depends on the specific scenario. In two-dimension routing, we use a similar Longest Prefix Match strategy to make the decision, which is described in detail in [2]. In packet classification and Openflow, we select the action of entry that has the highest priority. Obviously, we never change the action of each point in the multidimensional space.

D. Combing the Cells

After the multidimensional space is divided into cells, we combine the cells to obtain a smaller set of boxes in two ways. One of them is to merge two adjacent cells that have the same action into one, like Fig.2. The other way is to use a big box to cover most cells and use few small boxes to cover the area whose action is different from the big box. In the combination procedure, we calculate the action that occurs most frequently in the certain area; combine the cells that are labeled with the action; reserve the cells that have different actions or combine these cells iteratively. In Fig.4, the area of (1*, *) is covered

by three cells that are labeled with the same action of 2, and we can combine these cells into one rectangle of (1*, *, 2).

IV. MDTC Algorithms

A. Binary Tree Based Implementation

In order to implement the MDTC, we need a proper data structure. We use the binary tree to complete the algorithm. Each intermediate node corresponds to a Space-box; each leaf corresponds to a cell and is labeled with action; the edge between two nodes is labeled with the dividing information, and the path from the root to current node is the prefix expression of the corresponding Space-box. Besides, we add a RBS for each node in favor of the computation. In general, a cell is equal to a leaf and a Space-box is equal to a node.

We implement the compression with two parts: dividing the multidimensional space into a series of cells and combining the cells.

B. Construction of Binary Tree

In the first part, we will create a binary tree top-down and divide the multidimensional space into cells during the procedure. In other words, the binary tree construction and space division are proceeding concurrently. The second part calculation is based on the binary tree. We will implement the two parts respectively.

When dividing a Space-box, we take the two heuristic ideas into account. During the division, the corresponding node will add two child nodes and the Space-box is split into two areas. Simultaneously, we calculate the RBS for two new nodes. Here is the dividing procedure:

(1) At the beginning, we have a Space-box that has a RBS containing all the Related-boxes in the original table and corresponds to the root node. The value of each field of the space is wildcard.

(2) Verify whether a node is a cell. As every node has a RBS, we scan the Related-boxes in it. If each Related-box contains the node completely, the node is a cell; else, it is not.

(3) If a node is not a cell, we calculate the NOC and NUP for each node (or Space-box) with the favor of RBS, and choose the proper dimension to make the division; else, we choose a proper action for the leaf.

Fig.7 shows the dividing procedure of the Fig.4. The binary tree begins with a root node of (*, *). The root contains all the entries in the original table. In the first binary tree, we divide the whole multidimensional space into two areas which correspond to the two child nodes. NOC of source dimension is 1 and NOC of destination dimension is 2, so we divide the space by source dimension. During the division, we calculate the RBS for the two new nodes and the elements of the RBS are all inherited from the root. In the second binary tree, as the node of (0*, *) satisfies the conditions of cell, we don't split the space and select action 1 for it. As the Space-box (1*, *) is not a cell, we divide the area iteratively. We continue to divide the Space-boxes that do not satisfy the conditions of cell. The third binary tree is the final tree. Corresponding to the graphic, the cells of all these leaves cover the whole multidimensional space totally and every point is covered exactly by one cell, as shown in Fig.4.

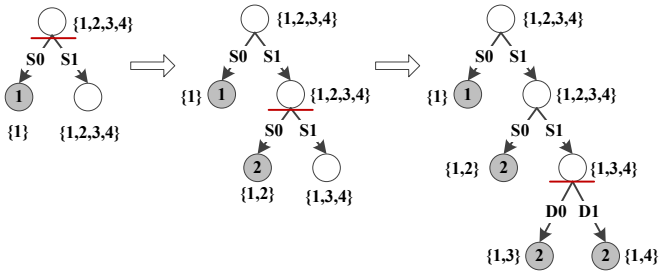

Fig. 7. The procedure of dividing. The node with underline is the split node; the leaf node is marked as grey; the RBS is recorded beside each node.

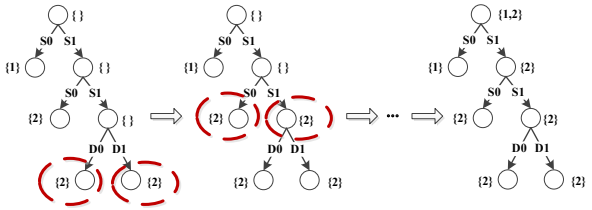

Fig. 8. Calculate the prevalent actions for each subtree from bottom to top. At the beginning, each leaf has a set of prevalent actions and the prevalent actions set of intermediate node is empty. We consider two prevalent action sets of the child nodes each time to calculate the prevalent actions for their father node.

After the completeness of dividing, the action information of each point is saved in leaves completely and it is not necessary for the intermediate nodes to keep their own actions. At the beginning of part two, the binary tree keeps the multidimensional table information in its leaves totally; all the nodes keep a RBS. And we use this binary tree as a baseline binary tree, for the sub-sequent operations are all based on it, like combination and update.

C. Combination

In the second part, MDTC travels from bottom to top in the binary tree to calculate the prevalent actions of each subtree; then, we select a proper action for each node from top to bottom. The operation is similar to ORTC [7]. During the calculation of prevalent actions, we consider two kinds of nodes:

(1) If a node is a leaf, its action is the prevalent one of the subtree rooted on it obviously.

(2) If a node is not a leaf, its prevalent actions are determined by its children. When the two children have the same prevalent actions, the intersection is the prevalent actions set of the subtree rooted in the node. Otherwise, the union is the prevalent actions set.

Upon finishing the calculation, every subtree has a set of prevalent actions, which are saved in the root of subtree. Fig.8 shows the prevalent actions calculation after division.

After the calculation of prevalent actions, the action of each node is empty. And then MDTC moves down the tree, picking up a proper action for each node. At the beginning, the root picks one action from its prevalent actions set. Next we deal with the nodes that have parent node. The node inherits an

action from its nearest ancestor, which has been visited, and then compares it with its own prevalent actions set. If the inherited action belongs to the set, then the node does not need to save the action because packets destined to this node will match the ancestor and will be delivered correctly; otherwise, it chooses an action from its prevalent actions set. In this way, the action information of some descendants are aggregated into their ancestors.

In the example, after calculating the prevalent actions, the root node is labeled with prevalent actions set $\{1,2\}$. MDTC chooses action 1 for the root randomly. The action 1 belongs to prevalent actions set $\{1\}$ of the left child, so the left child node does not save an action. However, action 1 is not a member of $\{2\}$, so the right child picks an action from its prevalent actions set and saves it. Fig.9 shows the top-bottom selection procedure and results with two non-empty nodes.

D. Construction of Compacted Table

Finally, we convert the compacted tree to a table. MDTC travels down the tree and records the path. When meeting a node labeled with action, we write the path and the action in the output table. After compression, every field is presented by prefix, which means we can save the compressed table in TCAM directly. Compared to range based compression algorithms, which need to convert the ranges to prefixes and are faced with the range expansion problem, MDTC compresses TCAM-based table better.

E. Correctness and Packets Handling

There are two significant questions: how to deal with the packets using the compacted table and how to guarantee the correctness. Firstly, we need to know how the router works with the compacted table. The output of our algorithm, the compacted table, is a series of entries whose fields are all defined as prefixes. We sum up all the prefix length of each field as the length of the entry. These entries are stored in TCAM. And the TCAM hardware compares the packet head with all the entries in parallel and returns the entry whose length is biggest, which is similar to FIB lookup. Corresponding to geometric representation, we will choose the smallest box to deal with the packet.

Then, let's consider the correctness problem. We achieve a set of cells firstly, and we select a proper action for each cell according to the different principle in different tables. Namely, we have guaranteed the correctness of each cell obtained. Then we combine several cells into a big box in order to reduce the number of entries. If all the points in the big box have the same action, the combination won't challenge the correctness obviously; if some cells have different actions from the big box, then we will keep these special cells and deal with the packets that match these cells correctly. In this way, the router deals with the packets in the same way before and after compression.

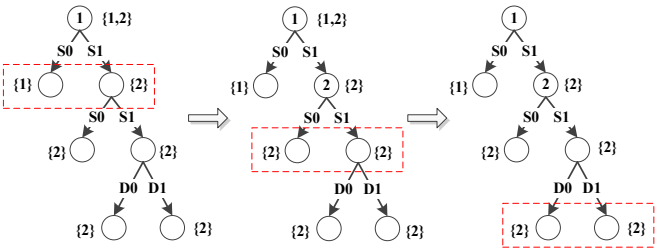

Fig. 9. Select an action for each node top-down, and the nonempty nodes are the result of output.

V. Experimental Results

A. Data Set

We conduct a series of experiments on real-life ACLs and generated multidimensional tables, like flow tables of Openflow, two-dimension tables and generated ACLs, and we define the size of table as the number of the entries. (1) ACL. We have 30 real-life ACLs from a real network (Because of the Dual and Double-blind Submission policy, we do not give the name of the network.). As many of them have fewer than 100 rules and have similar results, we only show the results of 8 ACLs whose size ranges from 20 to 1491. As it is difficult to obtain large number of real-life ACLs, we use Filter Set Generator [3] to generate a series of ACLs whose size ranges from 907 to 958. (2) Flow table. Based on the work of [14], we use the Stanford topology to generate Openflow flow tables whose size ranges from 2,063 to 38,804. (3) Two-dimension routing table. We use the real-life IP prefixes to generate two-dimension tables whose size ranges from 100 to 1,000,000. We obtain one-dimension BGP table in AS6447 and AS65000 from routeviews [15]. Then we use the table to generate two-dimension routing tables. In short, we have four experiment data sets: real-life ACLs, generated ACLs, generated flow tables and generated two-dimension tables.

B. Metrics

In this section, we evaluate the performance of MDTC by using the following three performance metrics: (1) Expansion ratio: $Expansion\ ratio = \frac{N_f}{N}$, where N denotes the number of entries in the original table with range-specific fields and N_f denotes the number of entries in the compacted table whose fields are all presented by prefixes; (2) Compression ratio: $Compression\ ratio = \frac{N_p - N_f}{N_p}$, where N_p denotes the number of entries in the original TCAM-compatible table and if the original table has range-specific field we will convert the field to prefix-specific field. (3) Time cost: the time to compress the table.

We compare our algorithm with Redundancy Removal [16] and Diplomat [13]. Redundancy Removal [16] is an excellent compression algorithm for its fast calculating speed and high compression ratio. Specially, Diplomat [13] and TCAM Razor [11] have adopted Redundancy Removal as part of their algorithm, and Redundancy Removal makes a significant improvement for them. So we believe it is meaningful to implement Redundancy Removal and compare our algorithm to it. As Diplomat is the latest algorithm for compression, which

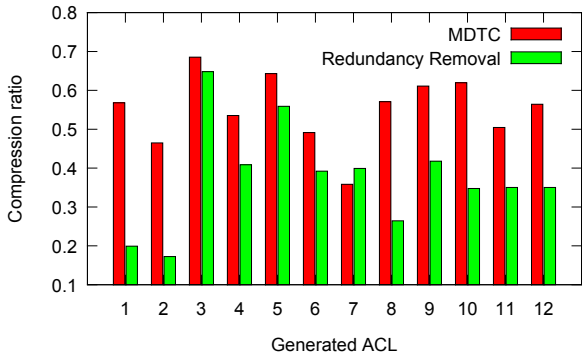

Fig. 10. The compression ratio of MDTC and Redundancy Removal.

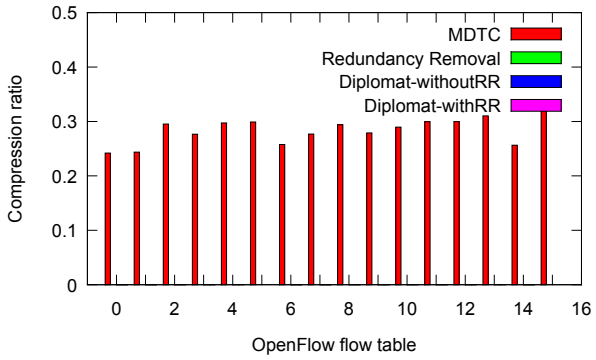

Fig. 11. The compression ratio of MDTC, Redundancy Removal, Diplomat-withoutRR and Diplomat-withRR.

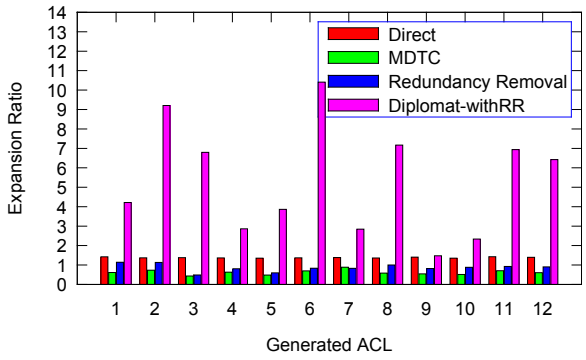

Fig. 12. The expansion ratio of Direct, MDTC, Redundancy Removal and Diplomat-withRR.

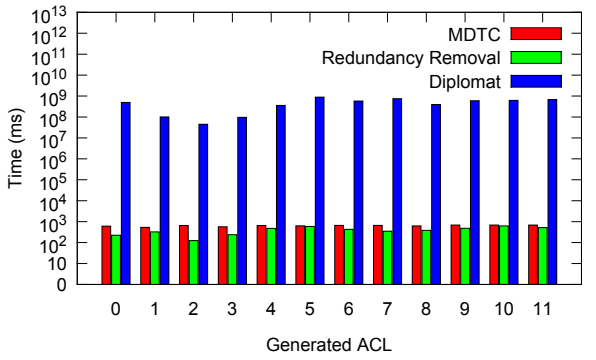

Fig. 13. The computing time cost of MDTC, Redundancy Removal and Diplomat for generated ACLs.

is outstanding for high compression ratio, we also compare our algorithm with its two versions: Diplomat-withRR and Diplomat-withoutRR, where Diplomat-withRR denotes the version of Diplomat with Redundancy Removal improvement and Diplomat-withoutRR is opposite. And Diplomat-withRR and Diplomat-withoutRR can be called Diplomat generally. We compare the output of these three algorithms with TCAM-compatible format.

The three algorithms have been implemented to compress the real-life ACLs, generated ACLs and generated flow tables. As the two-dimension routing table has a different matching scheme from ACLs, which is close to Longest Prefix Match, we do not use Redundancy Removal or Diplomat to compress them. Because Redundancy Removal and Diplomat are designed to compress ACLs.

C. Results

Compression Ratio. Fig.10 shows the results on generated ACls. The figure only shows the result of MDTC and Redundancy Removal. Though Diplomat compresses the table efficiently with range-specific format, the number of rules increases when we convert the output to prefix-specific format. As a result, the compression ratio of Diplomat is negative. MDTC has a compression ratio mean of 55.1%; Redundancy Removal has a compression ratio 37.6% averagely. MDTC has improved the compression ratio by 46.7% compared to Redundancy Removal. Fig.11 is the compression result on flow tables. We can see that MDTC reduces the tables by 24.2% at least and the compression ratio mean is 28.4%.

ACL	# of rules	MDTC	Redundancy Removal	Diplomat-withRR	Diplomat-withoutRR
ACL1	19	6	19	8	8
ACL2	39	37	39	37	37
ACL3	42	19	42	20	20
ACL4	52	52	52	52	52
ACL5	175	167	175	350	162
ACL6	663	515	663	-	-
ACL7	942	881	941	-	-
ACL8	1491	1363	1480	-	-

Fig. 14. The number of rules in real-life ACLs and the compacted ACLs.

On the contrary, Redundancy Removal and Diplomat almost do not reduce the table. Fig.14 shows the result of real-life ACLs. As it takes too much time for Diplomat to compress the ACL6, ACL7 and ACL8, we don't take these three results into account. For the real-life ACLs, MDTC obtains the smallest table compared to other algorithms. We also apply MDTC to two-dimension table compression and have a promising result that the compression ratio ranges from 25.82% to 55.10%. In summary, MDTC is able to compress any-dimension table efficiently.

Expansion ratio. ACLs have the problem of range expansion when we convert range rules to prefix rules for TCAMs. We calculate the expansion ratio mainly on the generated ACls, because the source port field and the destination port field are both range specified in the generated ACLs. Except for the three algorithms, we also calculate the expansion ratio when we change the range specified rules into prefix rules directly,

ACL	MDTC(ms)	RR(ms)	Diplomat(ms)
ACL1	1	1	4,109
ACL2	4	3	1,127,523
ACL3	1	2	11,000
ACL4	6	5	14,496,829
ACL5	27	14	36,051,712
ACL6	53	118	-
ACL7	142	554	-
ACL8	350	1187	-

Fig. 15. The time to compress ACLs.

shown as Direct in Fig.12. As shown in Fig.12, MDTC has the smallest expansion ratio mean of 0.62, which really decreases the original table; the Redundancy Removal gets the second smallest expansion ratio mean of 0.86; the Direct expansion ratio is 1.39 averagely. Though the Diplomat can compress the table into range specified table sufficiently, when we convert the output to prefix specified table the result expands greatly. We can learn from the Fig.12 that Diplomat-withRR gets the highest expansion ratio of 5.3 averagely. By comparison, we can conclude that MDTC reduces the table indeed for TCAMs.

Computing time. We calculate the computing time for real-life ACLs, generated ACLs and flow tables. As shown in Fig.15, MDTC spends several milliseconds to compress the small table and even uses less than one second to compress the largest ACLs with 1,491 rules. MDTC has a faster speed than Redundancy Removal when the size of table is more than 175. Diplomat has an enormous overhead in computing time and it even takes several hours for Diplomat to finish the compression. As shown in Fig.13, we find that MDTC and Redundancy Removal almost have the same computing speed to compress the generated ACLs and Diplomat still costs too much in computing time. We have also compared the time cost for flow tables, and acquire a similar result. In consequence, MDTC is fastest to compress the real-life ACLs, and is as quick as Redundancy Removal to compress the generated ACLs and flow tables. Compared to Diplomat, the time overhead of MDTC is very low.

In conclusion, MDTC compresses any-dimension TCAM-based tables efficiently with low cost in computing time. Specially, MDTC has an improvement of compression ratio for the generated ACLs by 46.7% compared to Redundancy Removal. Though Diplomat has excellent result of range specified compacted tables, the table expands when the range specified rules are converted to prefix specified rules according to the range expansion problem. So, we think that MDTC is the most appropriate algorithm for TCAM-based tables compression.

VI. Discussion

A. Lookup schemes

There are two types of lookup schemes: hardware-based and software-based. The compacted table, the result of our algorithm, is coded in ternary format and is available to TCAM lookup. On the other hand, the compacted binary tree, the intermediate product of our algorithm, provides an alternative software-based lookup scheme. When one packet arrives, router checks the information of packet head along the binary tree.

B. Updates

Multidimensional table is not invariable. There are three kinds of update: inserting new entries, modifying existing entries and deleting existing entries. In fact, we keep the baseline tree in memory preparing for the update and the baseline binary tree saves the table information in its leaves by the attribute of invisible-action. According to the principle of the longest prefix match, we guarantee the correctness by remaining the correctness for each cell. Generally speaking, at first, we look for the related leaves; then, we deal with the leaves and the subtree that contains these leaves further.

Insertion. When a new entry is inserted in the table, the box that the new entry defines must already has been set with one or more actions (different parts of the box may have different actions). We complete the insertion by two steps: finding out the related leaves (or cells) and amending the compacted tree.

The space the new box covers may have been split into several cells. To find these cells, we traverse the binary tree from top to bottom. After seeking out the related cells, we add the new entry to the RBS of each related cell. Then for each cell, we consider whether the box of the new entry covers the cell totally. If the box covers the cell totally, then the cell is still a cell. Then we check whether the action of the new entry is the same to the invisible-action. If they are the same, then nothing needs to be done; otherwise, we choose a proper action from the updated RBS. If the invisible-action is changed, we have to set the action for the cell no matter if it is labeled with an action originally. On the other hand, if the box only covers the cell partially, then the cell is not a cell any more and we need to split it into a series of cells while the old cell becomes a pre-cell. For the new cells that are related to the new entry, we use the same method as the first circumstance; and for the other new cells that have no relation to the new entry, we just set an invisible-action for them and copy the RBS from the pre-cell to them excepting the new entry.

Modification and Deletion. Similar to insertion, the first step is to find out the influenced leaves and then handle these leaves further. For the modification, we only need to check if the action of the modified entry is chosen for the leaves. If it is chosen, then we change the invisible-action and save the action for the leaves; otherwise, we do not need to do any modification for the leaves. For the deletion, we travel down the tree and find out the nodes whose RBS contains the deleted entry. Then, we remove the deleted entry from the RBS of those nodes, including intermediate nodes and leaves. For the affected leaf, if the deleted entry is chosen, then we select a proper action from its RBS and amend its invisible-action; otherwise, we need no more operation.

VII. Related Work

One-dimension compression algorithms. ORTC[7], the most efficient compressing algorithm for FIB, contains three passes: normalizing the binary tree top-down, calculating the

prevalent next hops in each level from bottom to top and selecting a proper next hop for each node from top to bottom. Then, a series of methods have been proposed to deal with updates or give incremental algorithms in[17–19].

Two-dimension compression algorithms. Suri et al. interpreted the two-dimension routing table with rectangles and exploit dynamic programming to compress the rectangles in [8]. Yang et al. have proposed a method that compresses source and destination dimension respectively, and then compacts the forwarding table [2].

ACL compression algorithms . Liu and Gouda proposed the first algorithm to shrink ACLs by eliminating the redundancy rule [16]. In [16], the authors take both the upward redundant rules and downward redundant rules into account and remove them. And in 2008 Liu et al. find a more efficient redundant rule removal algorithm [10]. Dong et al. proposed a reduction scheme for range expansion by converting ranges to prefixes [12]. Meiners et al. applied the one-dimension compression algorithm to each fields in packet classifier, and combined all the solutions to obtain a smaller equivalent packet classifier [11]. Daly et al. proposed a difference solution called Diplomat by transforming higher dimensional target patterns into lower dimensional patterns [13]. Besides, Liu proposed a BCAM based compression method [20] which makes use of BCAM to store the classifiers reducing the CAM bits. Recently,Kogan et al. proposed a hybrid software and TCAM-based approach to balance the memory space and lookup time in [21].

VIII. Conclusion and Future Work

We have designed a general compression algorithm for TCAM-based multidimensional tables. We first propose MDTC, which constructs a binary tree by dividing boxes into cells iteratively, and then combines cells to obtain a smaller table. Then we conduct a series of experiments to validate the proposed algorithms.

MDTC has four significant contributions. Firstly, MDTC addresses the range expansion problem well for TCAM-based ACLs. Second, MDTC is available to any dimensional tables. It's significant for a router to apply the same compression algorithm to different tables in it, like FIB and ACLs. Third, MDTC compacts the table efficiently. MDTC has improved the compression ratio by 46.7% compared to Redundancy Removal for generated ACLs. Last but not least, MDTC completes compression with a low cost of computing time. The low cost impairs the demand on devices and is meaningful for the dynamic systems.

Though MDTC has the advantages as mentioned above, it also has some limitations. The compression effect depends on the input, as we can see from the experiments that compression ratio on different data sets varies. Besides, when update comes we may create new entries, so the table grows as more and more updates occur. We will study on the two limitations in the future and continue to complete MDTC.

References

[1] N. McKeown, "Software-defined networking," in *INFOCOM keynote talk*, 2009.

[2] S. Yang, M. Xu, D. Wang, G. Bayzelon, and J. Wu, "Scalable forwarding tables for supporting flexible policies in enterprise networks," in *Proceedings of IEEE INFOCOM*, Toronto, Canada, Apr. 2014.

[3] D. E.Taylor and J. S.Turner, "Classbench: A packet classification benchmark," in *Proceedings of IEEE INFOCOM*, Miami, USA, Mar. 2005.

[4] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: enabling innovation in campus networks," in *Proceedings of ACM SIGCOMM*, Seattle, USA, Apr. 2008.

[5] A. Voellmy, J. Wang, Y. R. Yang, B. Ford, and P. Hudak, "Maple: Simplifying sdn programming using algorithmic policies," in *proceedings of ACM SIGCOMM*, Hong Kong, China, Aug. 2013.

[6] http://www.routeviews.org.

[7] R. Draves, C. King, S. Venkatachary, and B. Zill, "Constructing optimal IP routing tables," in *Proceedings of IEEE INFOCOM*, New York, USA, Mar. 1999.

[8] S. Suri, T. Sandholm, and P. Warkhede, "Compressing Two-Dimensional Routing Tables," *Algorithmica*, Apr. 2003.

[9] D. A. Applegate, G. Calinescu, D. S. Johnson, H. Karloff, K. Ligett, and J. Wang, "Compressing rectilinear pictures and minimizing access control lists," in *Proceedings of ACM-SIAM SODA*, 2007.

[10] A. X. Liu, C. R. Meiners, and Y. Zhou, "All-match based complete redundancy removal for packet classifiers in tcams," in *Proceedings of IEEE INFOCOM*, Phoenix, USA, Apr. 2008.

[11] C. R. Meiners, A. X. Liu, and E. Torng, "Tcam razor: A systematic approach towards minimizing packet classifiers in tcams," in *Proceedings of IEEE ICNP*, Beijing, China, Oct. 2007.

[12] Q. Dong, S. Banerjee, J. Wang, D. Agrawal, and A. Shukla, "Packet classifiers in ternary cams can be smaller," in *Proceedings of ACM Sigmetrics*, Saint-Malo, France, Jun. 2006.

[13] J. Daly, A. X. Liu, and E. Torng, "A difference resolution approach to compressing access control lists," in *Proceedings of IEEE INFOCOM*, Turin, Italy, Apr. 2013.

[14] P. Kazemian, M. Chang, H. Zeng, G. Varghese, N. McKeown, and S. Whyte, "Real time network policy checking using header space analysis," in *NSDI*, Berkeley, USA, Apr. 2013.

[15] http://www.routeviews.org.

[16] A. X. Liu and M. G. Gouda, "Complete Redundancy Detection in Firewalls," in *Proceedings of 19th Annual IFIP Conference on Data and Applications Security, LNCS 3654, pages 196-209*, Aug. 2005.

[17] Z. A. Uzmi, M. Nebel, A. Tariq, S. Jawad, R. Chen, A. Shaikh, J. Wang, and P. Francis, "Smalta: practical and near-optimal fib aggregation," in *Proceedings of the Seventh COnference on emerging Networking EXperiments and Technologies*, Tokyo, Japan, Dec. 2011.

[18] X. Zhao, Y. Liu, L. Wang, and B. Zhang, "On the aggregatability of router forwarding tables," in *Proceedings of IEEE INFOCOM*, San Diego, USA, Mar. 2010.

[19] Q. Li, D. Wang, M. Xu, and J. Yang, "On the scalability of router forwarding tables: Nexthop-selectable fib aggregation," in *Proceedings of IEEE INFOCOM*, Shanghai, China, Apr. 2011.

[20] A. X. Liu, C. R. Meiners, and E. Torng, "Packet classification using binary content addressable memory," in *Proceedings of IEEE INFOCOM*, Toronto, Canada, Apr. 2014.

[21] K. Kogan, S. Nikolenko, O. Rottenstreich, and W. Culhane, "Sax-pac (scalable and expressive packet classification)," in *proceedings of ACM SIGCOMM*, Chicago, USA, Aug. 2014.