Unified nvTCAM and sTCAM Architecture for Improving Packet Matching Performance

Xianzhong Ding
Shandong University, China
contactdingxz@gmail.com

Zhiyong Zhang
Shandong University, China
zhangzhiyongschool@163.com

Zhiping Jia *
Shandong University, China
jzp@sdu.edu.cn

Lei Ju
Shandong University, China
julei@sdu.edu.cn

Mengying Zhao
Shandong University, China
zhaomengying@sdu.edu.cn

Huawei Huang
The University of Aizu, Japan
davyhwang.cug@gmail.com

Abstract

Software-Defined Networking (SDN) allows controlling applications to install fine-grained forwarding policies in the underlying switches. Ternary Content Addressable Memory (TCAM) enables fast lookups in hardware switches with flexible wildcard rule patterns. However, the performance of packet processing is severely constrained by the capacity of TCAM, which aggravates the processing burden and latency issues. In this paper, we propose a hybrid TCAM architecture which consists of NVM-based TCAM (nvTCAM) and SRAM-based TCAM (sTCAM), utilizing nvTCAM to cache the most popular rules to improve cache-hit-ratio while relying on a very small-size sTCAM to handle cache-miss traffic to effectively decrease update latency. Considering the special rule dependency, we present an efficient Rule Migration Replacement (RMR) policy to make full utilization of both nvTCAM and sTCAM to obtain better performance. Experimental results show that the proposed architecture outperforms current TCAM architectures.

CCS Concepts • **Networks** → **Network architectures**; • **Hardware** → **Non-volatile memory**

Keywords TCAM, non-volatile memory, packet matching, Software-Defined Networking

1. Introduction

Software-Defined Network (SDN) is an emerging network paradigm that simplifies network management by decoupling the control plane and data plane, where switches act as data forwarding devices and network management is controlled by logically centralized servers (McKeown et al. 2008; Casado et al. 2009; Kang et al. 2013). By shifting the control plane to a logically centralized controller, SDN offers programmable functions to dynamically control and manage packets forwarding and processing in switches. The flexibility of SDN makes it easy to deploy a wide range of network management policies and new network technologies, e.g., traffic engineering (Benson et al. 2011), quality of service (QoS) (Ishimori et al. 2013), security/access control management (Gude et al. 2008), failure diagnosis (Kozat et al. 2014) and failover mechanisms (Sharma et al. 2013; Huang et al. 2016a).

In SDN, network flow is managed by a set of associated rules that are maintained by switches in their local Ternary Content Addressable Memories (TCAMs). A TCAM can compare a packet at line rate with all the patterns containing in the installed rules at the same time. However, commodity switches support relatively few rules, in thousands or tens of thousands (Stephens et al. 2012). On the other hand, fine-grained rich SDN policies result in tremendous amount of rules, which need to be stored and maintained in SDN switches. While TCAMs become indispensable due to their performance benefits, it comes with non-negligible cost. TCAM is power hungry, expensive and takes up quite a bit of silicon space (Huang et al. 2016b). The TCAM in modern commercial switch is based on SRAM (Kannan and Banerjee 2013). The high power leakage and low density of SRAM restrict the TCAM capacity significantly (Junsangsri et al. 2014; Guo et al. 2011; Zheng et al. 2014).

As the data amount generated from Internet keeps growing, to improve the TCAM capacity while minimizing the TCAM usage cost becomes a challenge. Recently, the non-volatile memory (NVM) such as resistive RAM (ReRAM) (Huang et al. 2014; Zheng et al. 2016) has been introduced to implement TCAM due to its merits of low leakage power and high density. Compared with the conventional sTCAM, NVM-based TCAM (nvTCAM) is reported to achieve higher storage capacity and lower energy consumption (Huang et al. 2014; Zheng et al. 2016). However, nvTCAM is not perfect due to its inherent high latency of write operation (Tsai et al. 2015). Furthermore, the update operation towards the installed rules in TCAM is a slow procedure. Today's SRAM-based switches only support around 40 to 50 rule-table updates per second (Jin et al. 2014; Huang et al. 2013). The update performance will be more serious for nvTCAM-based switches, highly constraining the network performance in large networks. To alleviate the limited capacity of sTCAM and the high write latency of nvTCAM, we propose a hybrid TCAM architecture which consists of both nvTCAM and sTCAM to take advantage of both high-capacity of nvTCAM to hold more high-contribution rules and a very small size of sTCAM to cope with newly arrived rules to reduce update latency.

* Zhiping Jia is the corresponding author

LCTES'17, June 21–22, 2017, Barcelona, Spain
© 2017 ACM. 978-1-4503-5030-3/17/06...$15.00
http://dx.doi.org/10.1145/3078633.3081034

The Least-Recently Used (LRU) replacement policy is widely used to manage buffer cache, but it does not work well to cope with access patterns with weak locality, such as dynamic arriving traffic. Therefore, with LRU scheme, the dynamic arriving packets will frequently consult the controller to ask for installations of new rules, and also induce too many table-miss events (ope), which is unacceptable for nvTCAM due to its high write latency. Low Inter-reference Recency Set (LIRS) replacement policy (Jiang and Zhang 2002) effectively addresses the limits of LRU by using recency to evaluate Inter-Reference Recency (IRR) for making a replacement decision. However, this strategy is only suitable for the blocks without dependencies and thus does not fit TCAMs. Considering the unique rule dependency in TCAM, we present a Rule Migration Replacement policy to effectively make full utilization of both nvTCAM and sTCAM to obtain better performance. The main contributions of this paper can be summarized as follows:

- We propose a hybrid TCAM architecture to make full use of high density of nvTCAM and efficient access of sTCAM.
- Based on the proposed hybrid TCAM architecture, a novel rule-migration strategy is designed, called Rule Migration Replacement (RMR) policy, to improve the rule update performance.
- Considering the unique rule dependency, a Replacement Value algorithm is designed to choose the best rules to be evicted in both nvTCAM and sTCAM.
- To demonstrate the performance of the proposed hybrid TCAM architecture, we construct comprehensive experiments with synthetic routing policies. Experimental results show hybrid TCAM outperforms current TCAM architectures.

The rest of this paper is organized as follows. First, we review related work in Section 2. In Section 3, we analyze the problem of rule management in the TCAM of SDN switches. In Section 4, we describe the proposed architecture and algorithm in detail. Experimental results are reported in Section 5. We conclude this paper in Section 6.

2. Related Work

The contradiction between the increasing number of flow table entries and the limited size of TCAM has become a serious issue in both traditional networks and SDN networks, and has received a lot of research efforts (Curtis et al. 2011; Rétvári et al. 2013). Related studies on TCAM's packet-processing problem can be categorized as follows.

Compression of TCAM Rules: The TCAM Razor (Liu et al. 2007) compresses multi-dimensional packet classification rules to minimize the overall TCAM rule set, using decision-tree and multi-dimensional topological transformation. Dong et al. (Dong et al. 2007) propose a caching technique for ternary table rules by compressing rules for evolving flows. In general, these techniques that use compression to reduce the TCAM space fail to adapt to the rapid growth of increasing rules in SDN networks.

Rule Caching: There are some recent studies dealing with the limited rule space of switches in SDN community. For instance, DIFANE (Yu et al. 2010) advocates the caching of TCAM rules, but uses more TCAM-based middle layer switches to handle the cache missing issues (ope). Devoflow (Curtis et al. 2011) introduces an idea of rule cloning to reduce the traffic volume processed by TCAMs. The key point of the proposed approach is to enforce each match in the TCAM triggering the installation of an exact-match rules (stored in SRAMs), aiming to handle the remaining packets of such micro-flows. However, Devoflow does not address the limitations on the total size of TCAM in the forwarding devices. Katta et al. (Katta et al. 2016) propose a cover-set based approach to solve the rule dependency, by creating a number of new rules that can cover the dependency-related rules and sending the affected packets to the software-based switches. However, this method also cannot deal with the conflict between the increasing flow entries and the limited TCAM size in the forwarding devices.

Besides, several hybrid TCAM architectures (Ullah et al. 2012) (Shen et al. 2012) have been proposed to eliminate the disadvantages like low bit density and high cost per bit of TCAM. HP-TCAM (Ullah et al. 2012) emulates TCAM functionality with conventional SRAM, and logically dissects conventional TCAM table in a hybrid way (column-wise and row-wise) into TCAM subtables, which are then processed to be mapped to their corresponding SRAM memory units. Shen et al. (Shen et al. 2012) present a novel tree-based hybrid TCAM+SRAM solution for the purposes of both TCAM space reduction and power saving. The main idea is to construct a compact tree from the classification rule set, with each level of the tree containing information for a particular header field involved in packet comparison. Upon the construction of the tree, it will be effectively mapped to a set of TCAM blocks and a single SRAM array. These two works both consider the hybrid architectures with SRAM and TCAM, and focus more on the hardware design, which are quite different from the proposed hybrid TCAM architecture composed of both nvTCAM and sTCAM. Our study fundamentally tackles the problem of limited TCAM capacity by achieving both high density of nvTCAM and low write latency of sTCAM.

3. Challenge and Motivation

In this section, we analyze the problems of rule management in the capacity-limited TCAM of SDN switches.

3.1 Limited Rule Space in OpenFlow Switches

Generally in SDN switches, the rules are stored in TCAM. TCAM provides the constant query time, which guarantees the forwarding speed of the switches. However, TCAM is a scarce and expensive resource. Each query in the TCAM accesses all the memory space, which makes the TCAM extremely power-hungry. Therefore, the on-chip TCAM size is quite limited. OpenFlow rules in SDN are also more complex than forwarding rules in traditional IP routers. For example, in OpenFlow 1.5, there are 44 fields in one flow entry (unk 2011). The current commercial TCAM-implemented SDN switch (e.g., Centec V330 switch) can only support about 2500 OpenFlow entries in the TCAM. However, in the current core Internet, there are more than 600,000 entries in the routing table (bgp 2016). The property of multiple fields will make the number of rules even larger. Therefore, in order to keep more rules in the data plane, it is crucial to improve the capacity of TCAM in SDN switches.

3.2 Rule Installation: Proactive or Reactive Mode ?

In SDN switches, flow rules could be installed in flow table in two ways: reactive mode and proactive mode.

- *Reactive Flow Rule Installation*: When a new flow arrives, the switch searches the flow table. If no matching rule is found, the switch sends packet-in message to the controller. The controller will install the corresponding rules after receiving the message. The reactive mode is flexible for dynamic traffic control based on realtime network conditions.
- *Proactive Flow Rule Installation*: In this approach, instead of reacting to a new flow, an OpenFlow controller pre-populates the flow tables of switches before the network traffic arrives. New flows matched by the prepopulated rules can be handled without consulting the controller.

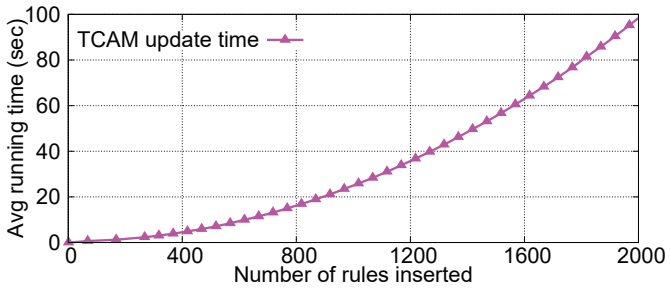

Figure 1. TCAM Update Time (Katta et al. 2016).

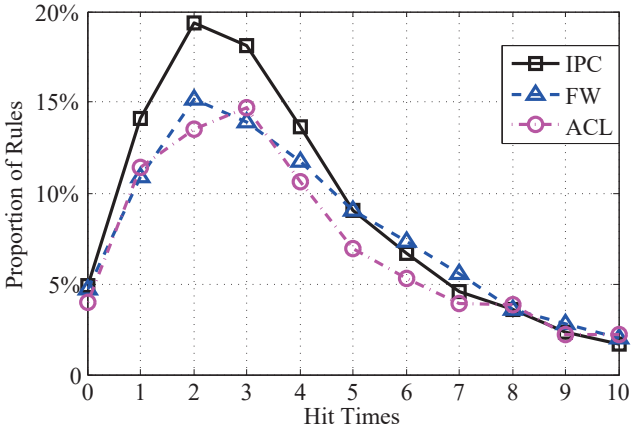

Figure 2. Hit Times of Rules.

Proactive mode and reactive mode both have their own advantages. On one hand, proactive mode and reactive mode are suitable for different applications: the former can guarantee the required bandwidth and latency for specific applications (e.g., virtual private network and enterprise applications), whereas the flexible reactive mode is employed to provide best-effort service, such as on-demand applications (e.g., online load balancing). On the other hand, in term of the essence of programming the network on the fly, the reactive rule installation is more flexible than the proactive one. For example, when new flows arrive, the reactive mode can direct these flows to less congested paths according to the current network condition. Some applications may require low latency, while others may need to be scheduled according to the current network situation. However, it is difficult to accurately predict which type of flows would arrive. Therefore, neither the reactive mode nor the proactive mode alone can satisfy the requirements of these diverse applications. A combination of these two modes can provide the flexibility of reactive mode and meanwhile preserve high-quality service for the specific traffic. In this paper, we discuss how to apply reactive and proactive rule installation approaches to nvTCAM and sTCAM, respectively.

3.3 Update Cost

In TCAM, as the number of entries increases, updates become more complex as they typically require re-ordering of existing entries (Rotsos et al. 2012). Therefore, the flow update cost is a big issue in TCAM, no matter in reactive mode or proactive mode. Take the rule insertion as an example, once a table miss occurs, the controller needs to insert a rule into the switch. However, the time required to update TCAM is non-linear to the rule number stored in it. As Figure 1 shows, the insertion of first 500 rules takes 6 seconds; the insertion of next 1500 rules takes almost 2 minutes. Besides, if the removed entry represents an active flow in the network, the switch has to send the subsequent packet of that flow to the controller. This creates a huge sudden load on the controller and may result in a significant drop in flow throughput (Huang et al. 2016a). Furthermore, when a flow entry is removed in the presence of wildcard rules, all its dependent rules with lower priority should be removed together, which further increases the update cost.

3.4 Flow Table Access Characteristics

We performed a group of analyses on three real filter sets provided by Internet Service Providers (ISPs), a network equipment vendor, and other researchers working in the field (Taylor and Turner 2007). The filter sets range from 68 to 4557 entries and utilize one of the following formats:

- Access Control List (ACL) - standard format for security, VPN, and NAT filters for firewalls and routers (enterprise, edge, and backbone).
- Firewall (FW) - proprietary format for specifying security filters for firewalls.
- IP Chain (IPC) - decision tree format for security, VPN, and NAT filters for software-based systems.

We then record the hit times of all rules. The experimental result is shown in Figure 2. It can be observed that most of rules are hit by 1-5 times for all three traffic traces, i.e., 57.16%, 60.78% and 74.26% for ACL, FW and IPC, respectively.

Apparently, the frequent update of rules using nvTCAM can result in unbearable high update delay due to the inherent high write latency of nvTCAM.

Normally, the forwarding table of the modern switch is composed of sTCAM. Although the density of sTCAM is lower than nvTCAM, but the write performance of sTCAM is better than nvTCAM. In order to take advantage of both high-capacity of nvTCAM and write performance of sTCAM, we are motivated to propose a hybrid TCAM architecture using both nvTCAM and sTCAM. Such hybrid TCAM architecture is referred to as hTCAM in this paper. It is a scalable and lightweight hybrid framework for rule installation and update in a two-layer TCAM. A fact should not be ignored that the look-up speed of both nvTCAM and sTCAM is in nanosecond (Huang et al. 2014). Therefore, the look-up speed of our proposed hybrid TCAM architecture will not be degraded comparing with the existing TCAM architecture.

4. Architecture and Algorithm

4.1 Overview of the Proposed Architecture

The proposed hTCAM uses a two-layer TCAM as the packet forwarding table, and brings two new functional modules to the existing OpenFlow infrastructure: (i) Proactive Rule Generator which is used to choose rules that can capture large volume of traffic to nvTCAM at the initial stage; (ii) Rule Migration & Replacement (RMR) Scheduler. There are two situations where RMR scheduler is triggered. One is when a packet cannot match any rules in nvTCAM or sTCAM, the RMR scheduler will install the missing rule to sTCAM. Another situation is the rules in sTCAM that are hit again by a coming packet, RMR scheduler will determine whether this rule needs to be upgraded into nvTCAM using the proposed

RMR policy. A conceptual design of architecture and two aforementioned modules are shown in Figure 3.

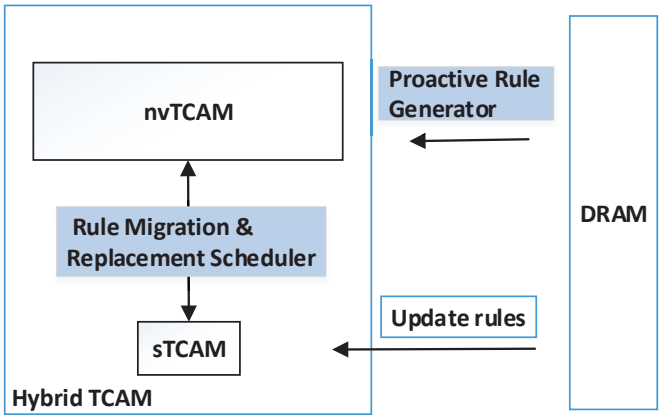

Figure 3. The proposed hybrid TCAM architecture.

4.2 Components in hTCAM

In the following, we depict each component of the proposed architecture shown in Figure 3.

4.2.1 A Two-layer Hybrid TCAM Rule Cache

As shown in Table 1, the search-latency and write-latency of nvTCAM are averagely 0.89× and 1.92× of sTCAM, respectively. Therefore, in the two-layer TCAM structure, we particularly divide the whole TCAM-based forwarding table into two layers according to their individual functions. The first layer is the ReRAM-based TCAM. This is a kind of nvTCAM that has been realized in (Huang et al. 2014), and is mainly used to hold the rules that can capture large volume of traffic. Then, the second layer is a small-capacity sTCAM (Huang et al. 2014) (Tsai et al. 2015). This layer is utilized to deal with the latest rules that miss or hit fewer times in order to reduce the update latency of hybrid TCAM.

Table 1. Comparison of sTCAM, nvTCAM (Huang et al. 2014) (Tsai et al. 2015).

		sTCAM	nvTCAM
Latency	Write	1×	1.92×
	Search	1×	0.89×
Silicon board area		1×	0.48×

1) *Proactive Rule Generator*: The *contribution value* of a rule R is defined as the ratio between its *weight* and its *cost*. Note that, we assume that each rule R is measured with a *cost* corresponding to the number of rules that must be installed together with it, and a *weight* corresponding to the number of packets expected to be hit. We can see that the *contribution value* can be used as a metric for rule caching in TCAM.

It is of much significance to generate high-contribution proactive rules covering as many packets as possible, which can greatly reduce rule cache miss and improve the packet forwarding rate. For rules that have disjoint patterns (e.g., micro rules with no wild-card bits), a greedy algorithm is capable to find a certain number of rules with high contribution. For rules with dependency-overlapped relations, the generation becomes complicated, because a high-contribution rule may be dependent with multiple other rules. The cover-set method which has been implemented by Katta et al. (Katta et al. 2016) effectively solves this dependency problem. The authors splice the dependency chain by creating a small number of new rules that cover many low-weight rules and send the affected packets to the software switch. Proactive Rule Generator still selects the high-contribution rules from DRAM (all the rules are stored in these memories) using cover-set method and aims to maximize the total contribution of the cached rules within the capacity constraint of nvTCAM. Therefore, the nvTCAM owning a larger capacity with the same silicon-board area is preferable.

A small-size sTCAM is essential, as long as its size can hold the rules as well as their dependent rules to perform update operations. Since the cost of rule update is different, once a cache-miss event occurs, the update time is twice longer than only inserting a rule when there is still some available space in the flow table (Jin et al. 2014). In order to further reduce the update latency, the proactive flow rule generator does not yield rules for caching in the sTCAM at the initial stage. Once a rule misses, the SDN controller will trigger the installation of a new rule as well as its cover-set rules for the first arriving packet. Then, all flow entries will be removed automatically once reaching the predefined timeout (hard timeout or idle timeout (ope)), which avoids some "dead" rules always occupying sTCAM but matching nothing. Once a packet arrives at a switch, it firstly tries to match with the rules installed in the nvTCAM. If it successfully matches a rule (or multiple rules), it will be processed by the corresponding actions indicated by the rule. Otherwise, it will be forwarded to sTCAM to perform further comparison. However, if this packet still fails to match a rule, it will trigger a flow setup request, which is sent to the SDN controller. Then, controller installs the missing rule along with its cover-set rules into the sTCAM.

2) *Rule Migration & Replacement Scheduler*: After caching proactive rules into nvTCAM or installing the missing rules into the sTCAM, the input packets can match rules either in nvTCAM or sTCAM.

In order to adapt to the dynamic network, it is necessary to design a flow table migration and replacement strategy considering the characteristics of the rules in nvTCAM or sTCAM. This motivates us to design new rule replacement strategies for the proposed hybrid TCAM. We introduce a novel Rule Migration Replacement (RMR) policy between the first-layer nvTCAM rule cache and the second-layer sTCAM rule cache. The detailed policy is described in the following sections.

4.3 Rule Migration & Replacement Algorithm

4.3.1 Idea of RMR

Here we present the RMR algorithm that serves the RMR scheduler. To describe this algorithm, some basic concepts are needed to specify. The rules installed in nvTCAM and sTCAM are called the Low Inter-reference Recency (LIR) rule set and High Inter-reference Recency (HIR) rule set, respectively. The recent Inter-Reference Recency (IRR) is used as the recorded history information of each rule, where IRR of a rule refers to the number of other rules accessed between two consecutive references to the rule. Specially, the *Recency* refers to the number of other rules accessed from last reference to the current time.

Furthermore, we call the IRR between the last and the penultimate (second-to-last) references of a rule as the *recent IRR*. It is assumed that if the IRR of a rule is large, the next IRR of the rule is likely to be large, too. In nvTCAM, the rules with large IRRs are to be selected preferentially for replacement. In comparison with LRU scheme, by adequately considering the historical IRR information, the replacement strategy under IRR can significantly degrade the negative effects caused by only considering recency. In summary,

applying such policy taking the IRR information into account, the rules with LIR are much likely to be cached in nvTCAM.

Algorithm 1: Rule Migration Replacement Algorithm

```
Input  : rules in sTCAM R_s, rules in nvTCAM R_n;
         missing rules R̂, rules hit in sTCAM R_hit;
Output : rules degraded into sTCAM R_deg;
         rules migrated into nvTCAM R_mig;
1  if isMiss(R̂) then
2      Each R_s in descending R_s.RV;
3      while freeSpace(sTCAM)<needSpace(R̂) do
4          delete(R_s);
5      Insert_sTCAM(R̂ + cover-set of R̂);
6  if isHit_sTCAM(R_hit) then
7      if R_hit.IRR < R_n.recency then
8          Each R_n in descending R_n.RV;
9          if R_hit.RV < R_n.RV then
10             while freeSpace(nvTCAM)<needSpace(R_hit)
               do
11                 put R_n in Queue;
12                 updateSpace(nvTCAM);
13         else
14             break;
15     R_s in descending R_s.RV;
16     while freeSpace(sTCAM)<needSpace(Queue) do
17         delete(R_s);
18     if Queue! =NULL then
19         R_deg=degrade_sTCAM(Queue);
           R_mig=upgrade_nvTCAM(R_hit+cover-set of R_hit);
```

As illustrated in Algorithm 1, when a rule-missing event occurs to a new rule $\hat{R}$, the RMR scheduler first checks whether the sTCAM has enough available table space for $\hat{R}$ and its related cover-set rules. If yes, the RMR scheduler directly inserts them to sTCAM. Otherwise, it decides to delete existing rules installed in sTCAM using replacement value algorithm until achieving the required table space for $\hat{R}$ and its cover-set (Lines 1-5). If R_{hit} in sTCAM is hit by any packet later, the RMR scheduler will determine whether this rule should be upgraded to nvTCAM. Firstly, RMR scheduler checks whether the IRR of R_{hit} is lower than the recency of rules kept in nvTCAM, and then the *Replacement Value* (RV) for each rule in nvTCAM is calculated. After that RMR scheduler checks whether the RV of R_{hit} is less than RV of rules in nvTCAM. If such conditions are satisfied, then R_{hit} and its related cover-set rules will be migrated into nvTCAM (Lines 7-14), and the rules that make space for R_{hit} and its cover-set rules will be degraded to sTCAM (Lines 15-19).

4.3.2 Quantify the Replacement Value of Each Rule

The replacement value is of much significance in the proposed architecture, because the high write latency of nvTCAM and the inherent update problem of traditional TCAM will degrade hybrid TCAM's rule match performance. Therefore, we rely on rule dependency characteristics to pick up the best rules in nvTCAM to replace in order to reduce the number of rule installation of nvTCAM. All the rules stored in nvTCAM can construct a dependency graph, where rules in different positions have different replacement values. For example, Figure 4 shows an overall dependency graph among all rules. The replacement value of each rule is calculated as follows.

$$R_i.RV = (\alpha \times \gamma + (1-\alpha) \times S)/PN_{R_i}, \quad (1)$$

where γ is the recency of rule R_i, which is the major factor when deciding to remove a rule; S refers to the space can be released when this rule is removed; α is the regulatory factor that decides the weight of the emphasizing factor (recency or space); PN is parent number of the rule. For example, if rule A depends on rule B, then B is the parent of A. As shown in Figure 4, R2, R3, R4 are the parents of R1, and in order to facilitate the calculation we will consider R1 itself as its own father, thus the parent number of R1 is 4. We consider the parent number because removing different rules will incur different times of insertion, the rules with small parent number are preferred to be selected for replacement.

Table 2. Replacement Value 1.

Rule	Recency	Space	ParentNumber	RV(α=0.6)
R1	5	1	4	0.85
R2	4	1	3	0.93
R3	3	1	2	1.1
R4	**2**	**2**	**1**	**2**
R7	1	3	1	1.8

Table 3. Replacement Value 2.

Rule	Recency	Space	ParentNumber	RV(α=0.6)
R1	5	1	3	1.133
R2	4	1	2	1.4
R3	**3**	**2**	**1**	**2.6**
R7	1	3	1	1.8

Here is an example to quantify the Replacement Value of a rule. Suppose the nvTCAM capacity is 6 and sTCAM is 2. Now R1, R2, R3, R4, R6*, R7 are in nvTCAM and R8, R9 are in sTCAM, the dependency graphs of the rules in nvTCAM and sTCAM are shown in Figure 4. No matter they are in nvTCAM or sTCAM, the rules in them always construct their own dependency graph. In Figure 5, the rules in grey color (R1, R2, R3, R4, R6*, R7) construct the dependency graph of nvTCAM and the rules in white color (R8, R9) construct the dependency graph of sTCAM. The recency of the rules in nvTCAM are in the descending order sequence. Now it's assumed that R9 reaches the conditions to migrate into nvTCAM from sTCAM. R9 and its child R8 need two spaces, so first we should choose some rules in nvTCAM to be removed and use Replacement Value algorithm to decide which rule need to be removed. As Table 2 shows, the biggest Replacement Value of all the rules in nvTCAM is R4, so RMR scheduler first deletes R4 and makes only one space. However R8 and R9 need two spaces, so RMR scheduler will calculate next biggest Replacement Value of rule as Table 3 shows after removing R4. The Parent Number of R1, R2 and R3 will change, so we will calculate new biggest Replacement Value rule R3, then after removing R3, the spaces that for R8 and R9 are met. First we need to remove R4 and R3 in nvTCAM, then insert R8 and R9. The removed rules R3 and R4 will be degraded into sTCAM after the migration.

We give an example to illustrate the working process of the hybrid TCAM architecture. We divide the whole process into three steps: (i) Initialization Phase: It is assumed that the capacity of nvTCAM and sTCAM is 6 and 2, respectively. There are 9 rules in DRAM, proactive rule generator first chooses 6 highest contribution rules. We suppose R1, R2, R3, R4, R6* and R7 are the selected

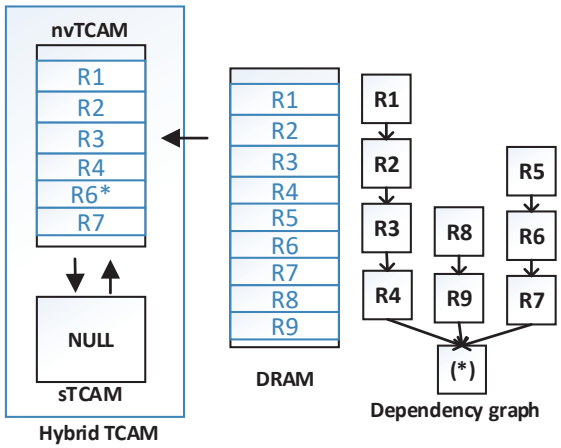

Figure 4. Initial rules of nvTCAM and sTCAM.

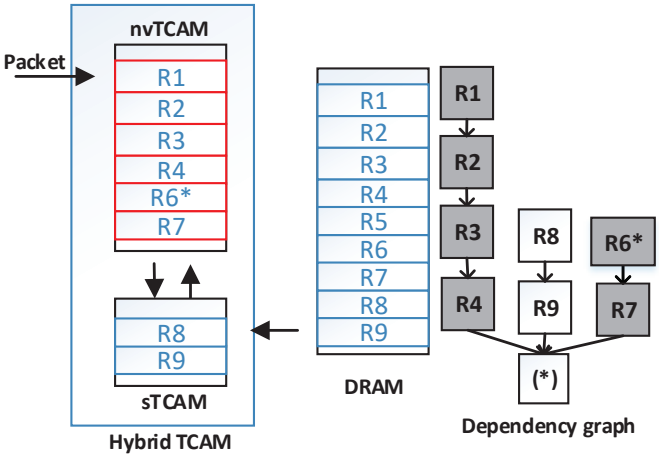

Figure 5. Packet Matching.

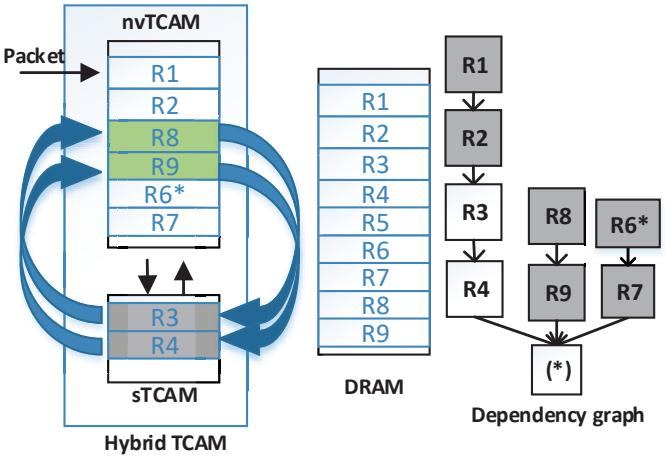

Figure 6. Replacement and Migration.

rules, and proactive rule generator installs these rules to nvTCAM as Figure 4 shows. No rule is chosen to sTCAM at this stage. After selection, the process proceeds to the second step; (ii) Packet Matching: Suppose the sequence that packets access the rules is R1, R2, R9, R9. First the packet visiting R1 comes. It first compares the rules in nvTCAM and successfully hits, then takes the corresponding action. Then, the packet visiting R2 comes and hits again in nvTCAM. But when the packet matching R9 comes, it first compares all rules in nvTCAM and sTCAM, and misses. Then a request is sent to SDN controller. RMR scheduler will first install the missing rule R9 and its dependent rule R8 to sTCAM. Then this packet hits. The dependency graphs in nvTCAM and sTCAM are the grey graph and white graph as Figure 5 shows. The last one is also the packet matching R9. It still compares to the rules in nvTCAM but misses, then it compares to the rules in sTCAM and hits. Then the third process is triggered; (iii) Replacement and Migration: RMR scheduler then determines whether the R9 in sTCAM needs to be migrated into the nvTCAM. If the condition is not met, R9 is still in sTCAM and waits for the next hit. But if R9 satisfies the migration condition, RMR scheduler calculates the space that R9 needs to migrate into nvTCAM is 2, then R4 and R3 are chosen by using the replacement value to make space. As Figure 6 shows, R8 and R9 migrate from sTCAM to nvTCAM, and the replaced rule R3 and R4 will be degraded into sTCAM, and the dependency graphs of nvTCAM and sTCAM are shown in Figure 6.

4.3.3 Complexity Analysis

In the worst case, Rule Migration Replace Algorithm runs in $O(n\log n)$, where n is the number of rules in nvTCAM, because we should sort the rules by their Replacement Values. For the most practical policies, the running time is $O(n)$, which is linear to the number of rules cached in nvTCAM.

5. Experiments

We implement a prototype of hybrid TCAM in Python based on the Ryu controller platform. Currently, this prototype transparently supports the semantics of the OpenFlow 1.0 (ope) features. We compose such prototype using a collection of three Open vSwitch (OVS) 2.1.2 instances where the first OVS instance serves as the hardware-cache that is viewed as nvTCAM, while the second instance serves as the sTCAM. The rule-table-size capacity of the two OVS instances is determined by the predefined capacity of hybrid TCAM. And the third one acts as the software switch that holds the cache-miss rules. Utilizing such prototype, we evaluate the performance of the proposed schemes with synthetic policies.

5.1 Simulation Setup

Rule-Generation Policy & Trace generation: Synthetic rule policies are generated by ClassBench (Taylor and Turner 2007), which is an open-source tool deployed in the real datacenter database to generate the synthetic rule policies with the desired different number of rules and the corresponding traffic traces. We generate a dataset that includes three standard-policy Access Control Lists (ACLs), three FireWalls (FWs) and two IP Chains (IPCs) and a certain number of traffic flow traces, from seed files provided by ClassBench. The dataset is shown in Table 4.

Parameter Settings: The proposed hybrid TCAM is particularly designed for the condition where the number of flow entries is greater than the total TCAM capacity volume (the total size of both sTCAM and nvTCAM). In our simulation, we compare pure sTCAM and nvTCAM with hybrid TCAM. Assuming the same physical area, we calculate the number of rules that pure sTCAM and nvTCAM can hold based on data listed in Table 1, which is 500 and 1040, respectively. The size of hybrid TCAM is set to

Table 4. Dataset.

Name	Number of Flow Entries	Number of Traces
ACL1	2785	28637
ACL2	2771	12561
ACL3	3825	14602
FW1	3471	22838
FW2	1267	10135
FW3	2536	21734
IPC1	5527	25865
IPC2	1507	28625

Table 5. Three Types of TCAM.

TCAM	sTCAM	nvTCAM	Hybrid TCAM	
			nvTCAM	sTCAM
Size	500	1040	936	50

986, where nvTCAM occupies 90% of the whole chip area and holds 936 rules, while the sTCAM part occupies 10% of the area and holds 50 rules. The ratio achieves the best performance for our hybrid TCAM according to simulation results as shown in Figure 9. We also evaluate the performance when applying the proposed RMR algorithm to the hybrid TCAM architecture.

5.2 Simulation Results

In this section, we choose cache hit rate and update latency as main metrics to test the proposed hybrid TCAM architecture, which are typically used as the performance metrics by the literature work to evaluate the TCAM design and/or rule caching scheme, e.g., (Katta et al. 2014, 2016; Sheu and Chuo 2016).

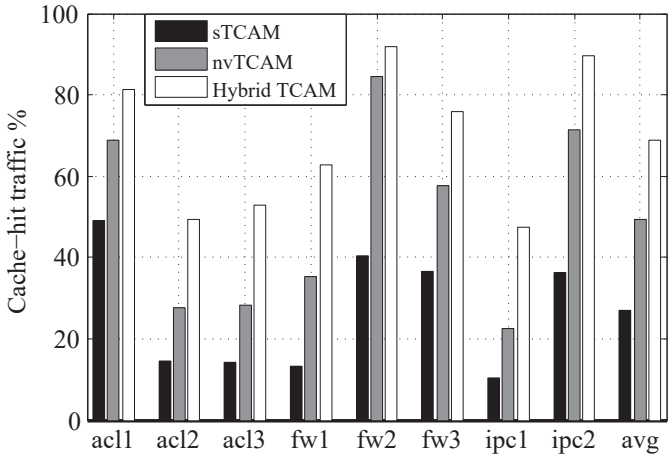

Figure 7. The performance of cache-hit under three types of TCAM.

5.2.1 Cache-hit Rate

In this group of experiment, we use eight seed files (three ACL seeds, three FW seeds and two IPC seeds) provided by ClassBench to generate different number of rules and corresponding packet traces to separately measure cache-hit rate and update latency under three types of TCAM. The specified rule-table size is shown in Table 5. The cache-hit results of eight datasets are demonstrated in Figure 7, where we can see that under all data sets, the hybrid TCAM with RMR policy averagely achieves a cache-hit rate by 41.97% higher than sTCAM with LRU replacement algorithm, and 19.41% higher than nvTCAM with LRU replacement algorithm. The reasons are as follows. (i) compared with sTCAM under LRU, hybrid TCAM consists of nvTCAM and sTCAM, which greatly increases the rule-table size of whole TCAM to hold more rules with high-contribution value; (ii) compared with pure nvTCAM, the capacity of pure nvTCAM is 5.19% higher than the hybrid TCAM. But under the LRU algorithm, rules with high contribution that are chosen at the initial stage in pure nvTCAM will be soon replaced by new coming rules. The new rules might be only visited by few times, which greatly reduce the cache-hit rate. However, the hybrid TCAM not only takes advantage of the high capacity of nvTCAM, but also employs a small capacity of sTCAM to update the new coming rules under the proposed RMR algorithm. When updating rules, the RMR algorithm takes special rule dependency into consideration instead of only the recency used by LRU, maintaining more high-contribution rules in nvTCAM instead of evicting them by the new coming rules.

Note that, in the hybrid TCAM, the area ratio between the nvTCAM and sTCAM is 9 to 1. Here we only use a very small-size sTCAM. When a rule misses, the missing rule and its cover-set rules should be chosen to install in sTCAM together. Since the number range of dependent rules is 1-20, the size of sTCAM with just a little bit more than the maximum number of cover-set rules is enough.

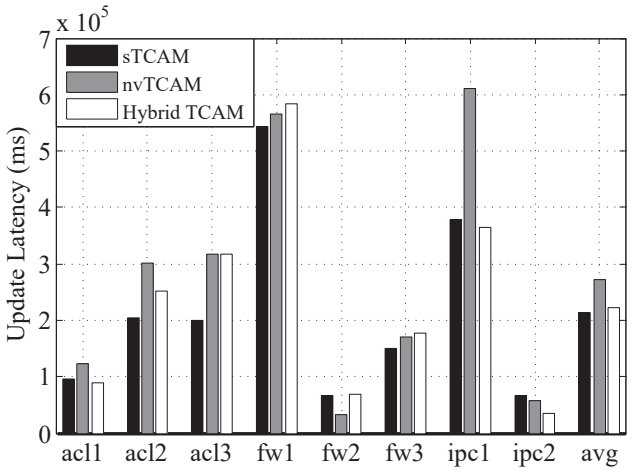

Figure 8. Update latency of three types of TCAM.

5.2.2 Update Latency

The update in TCAM is a very complicated task as mentioned previously. When a rule is installed in TCAM, its cover-set rules must be installed together to guarantee the semantic correctness of packet processing. The result is illustrated in Figure 8, where we observe that the update latency of pure nvTCAM is averagely 27.80% higher than pure sTCAM. This is because the write latency of nvTCAM is 1.92× higher than the sTCAM. However, the latency of the proposed hybrid TCAM is a little higher than pure sTCAM, with only 4.05% increasement on average. The reason can be explained as that we use the sTCAM as the second layer cache to update the newly arrived rules as well as their cover-set rules, and further use the value-based replacement scheme to effectively restrict the rules stored in sTCAM to migrate into nvTCAM. As a result, only the

rules with high contribution can be upgraded to nvTCAM, greatly reducing the write latency for the first layer of hybrid TCAM.

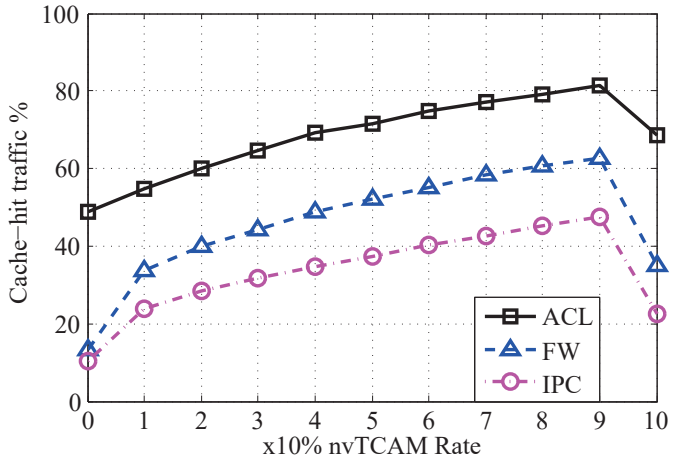

Figure 9. The effect of nvTCAM rate.

5.2.3 Different NVM Rates in Hybrid TCAM

We use ACL, FW, IPC data sets to explore the effect of different nvTCAM rates, which is defined as the area percentage of nvTCAM in the entire hybrid TCAM. As shown in Figure 9, the case that nvTCAM rate is equal to 0 means the hybrid TCAM is essentially the pure sTCAM. Thus it has the lowest cache-hit rate at the beginning. We can see that the cache-hit rate under all data sets increases as the nvTCAM rate increases before the nvTCAM rate reaches 90%, where the cache-hit ratio of all data sets reaches the peak. This is because nvTCAM rate has reached the maximum to hold the high-contribution rules and 10% of the area of sTCAM must be used to update the new rules. However, when nvTCAM rate reaches 100%, the cache-hit rate decreases. This is because the hybrid TCAM becomes pure nvTCAM, where the proposed RMR algorithm loses its advantage in rule migration between two TCAM layers. As a result, it cannot effectively avoid the rules with high-contribution value being replaced by newly arrived rules.

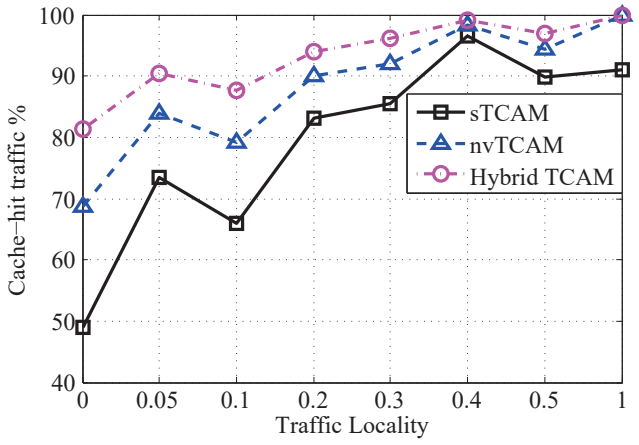

Figure 10. Locality of ACL.

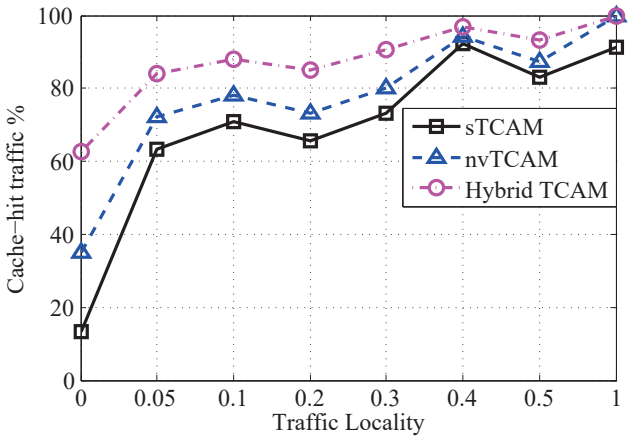

Figure 11. Locality of FW.

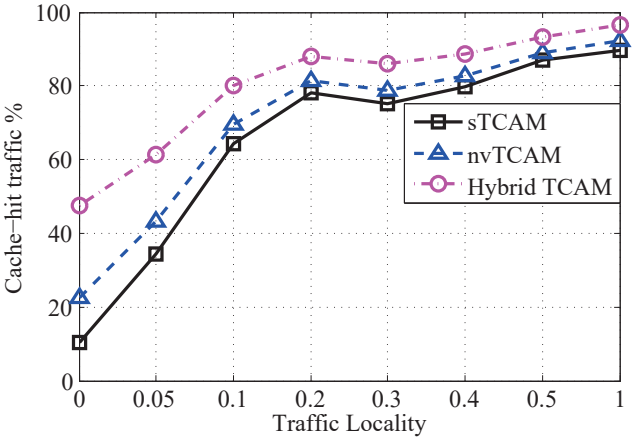

Figure 12. Locality of IPC.

5.2.4 Different Locality of Traces

We investigate the relation between the cache-hit rate of three types of TCAM from weak locality 0 to strong locality 1. Via the observation from Figure 10, Figure 11 and Figure 12, we find that ACL, FW and IPC have the same performance trend: the cache-hit of all three TCAMs increases with stronger locality. This is because with the increase in traffic locality, more packets hit the same rules in TCAM. In particular, the cache-hit of the nvTCAM is always higher than sTCAM. This is because the size of nvTCAM is larger than the size of sTCAM. We also observe that the cache-hit of hybrid TCAM is higher than that of both pure sTCAM and nvTCAM. The reason is that the sTCAM in hybrid TCAM handles the missed rules, and RMR scheduler further restricts the migration from the sTCAM to nvTCAM, making the first-layer TCAM always store the rules with high contribution.

5.2.5 Hybrid TCAM under LRU & RMR Policies

To illustrate the performance of the proposed RMR strategy, we conduct a set of experiments to show the advantage of RMR algorithm over traditional LRU policy for hybrid TCAM. We separately measure cache-hit rate and update latency under LRU and RMR

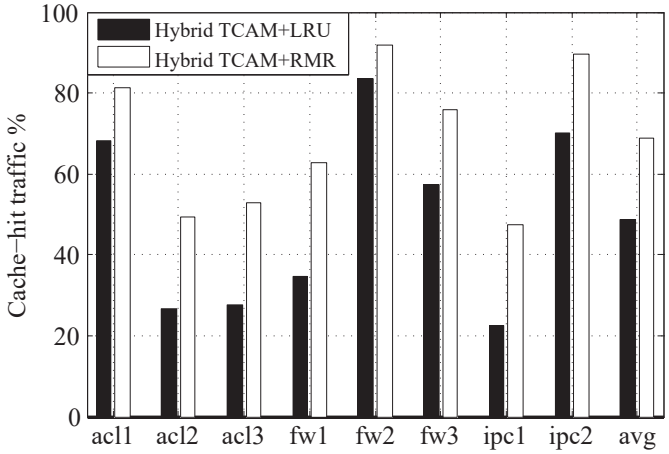

Figure 13. The cache-hit ratio of nvTCAM under LRU and RMR algorithm.

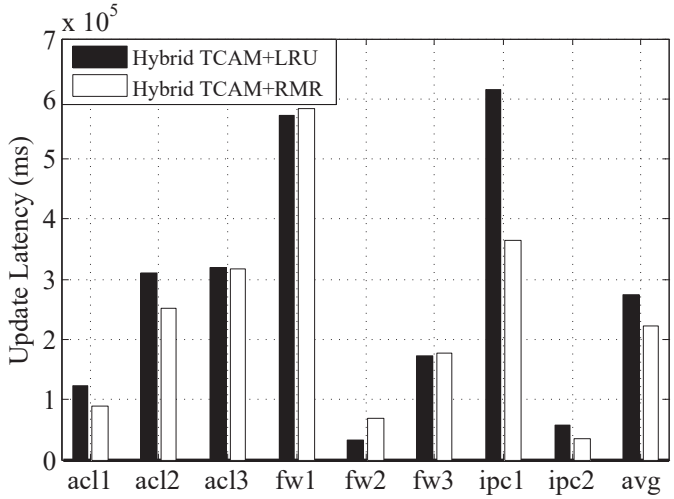

Figure 14. Update latency of nvTCAM under LRU and RMR algorithm.

strategies of hybrid TCAM. The cache-hit and update latency results of eight datasets are demonstrated in Figure 13 and 14. We can see that under all datasets, the hybrid TCAM with RMR algorithm averagely achieves a 20.08% higher cache hit rate and a 19.52% lower update latency than hybrid TCAM with LRU replacement algorithm. The reasons are as follows. (i) in comparison with hybrid TCAM under LRU, hybrid TCAM with RMR not only exploits the larger capacity of nvTCAM, but also uses a small-size sTCAM to handle the newly arrived rules. Our proposed RMR algorithm keeps more high-contribution rules in nvTCAM instead of evicting them and thus makes these rules keep hitting the coming packets. Instead, hybrid TCAM under LRU considers the nvTCAM and sTCAM as a unified TCAM. When replacing a rule in this unified TCAM, the LRU replacement algorithm inevitably evicts some of the high-contribution rules in nvTCAM and thus leads to a lower hit rate; (ii) as for the advantage of the update latency of RMR, we fully consider the advantages of nvTCAM and sTCAM according to the rule dependency characteristics. Due to the low write latency, sTCAM is used as the second layer cache to update the newly arrived rules, and the value-based replacement algorithm effectively restricts the rules stored in sTCAM to migrate into nvTCAM. As a result, RMR algorithm only upgrades the rules in sTCAM with high contribution to nvTCAM, thus greatly reducing the write latency for the nvTCAM. However, the LRU replacement algorithm only takes the recency of rules into account without considering the special rule dependency. Besides, LRU does not differentiate two layers in the hybrid TCAM. As a result, a large number of updates occur in the nvTCAM portion, inducing a much higher update latency.

Table 6. Write time of hybrid TCAM under LRU and RMR policy.

	nvTCAM	hTCAM+LRU		hTCAM+RMR	
	+ LRU	sTCAM	nvTCAM	sTCAM	nvTCAM
ACL1	22273	1087	21280	29915	1134
ACL2	53547	2690	52064	83713	1215
ACL3	56005	2766	53494	67797	1143
FW1	99186	4967	95334	194440	1242
FW2	6552	290	6450	22696	1089
FW3	30803	1506	29553	59162	1125
IPC1	107199	5334	102282	121792	1341
IPC2	11029	503	10501	11324	1071
AVG	48324.3	2392.9	46369.8	73854.9	1170

5.2.6 Lifetime of nvTCAM in Prue nvTCAM and Hybrid TCAM

The lifetime of nvTCAM is less than sTCAM, if the nvTCAM part of hybrid TCAM is used frequently, it can be worn out very early. We use eight data sets to demonstrate the write times of pure nvTCAM, hybrid TCAM under LRU replacement algorithm and hybrid TCAM under our proposed RMR algorithm, respectively.

As Table 6 shows, the average write time of pure nvTCAM with LRU algorithm is 48324.3. The average write time of nvTCAM in hybrid TCAM with LRU algorithm is 46369.8 which accounts for 95.09% of the whole hybrid TCAM, and is 4.04% lower than pure nvTCAM. This is because the capacity of nvTCAM accounts for 95% of the whole capacity of hybrid TCAM, and LRU algorithm only considers the recency when considering replacing a rule which inevitably results in the high write times of the big-size nvTCAM. This is unbearable for the limited endurance of nvTCAM. However, under the proposed RMR algorithm, the average write time of nvTCAM in hybrid TCAM is 1170 which only accounts for 1.56% of the whole hybrid TCAM, and is 97.58% and 97.47% lower than pure nvTCAM and hybrid TCAM under LRU algorithm. This is because all the missing rules will insert into sTCAM first. Besides, when replacing a rule, we take special rule replacement value into consideration, and only those high-contribution rules can migrate into nvTCAM. The proposed RMR algorithm makes more than 98.44% of the write time happen in sTCAM, which greatly improves the lifetime of nvTCAM.

6. Conclusion

In this paper, we propose a hybrid TCAM architecture based on the conventional NVM-based TCAM and SRAM-based TCAM, utilizing nvTCAM to cache the most popular rules to improve cache-hit-ratio while relying on a very small-size sTCAM to handle cache-miss traffic to effectively decrease update latency. Considering the special rule dependency, we then devise a rule migration replacement algorithm to effectively coordinate both nvTCAM and sTCAM to obtain better performance. The experimental results confirm that the proposed architecture outperforms the current TCAM architectures significantly.

Acknowledgments

This research is sponsored by the State Key Program of National Natural Science Foundation of China No. 61533011, National High-tech R&D Program of China (863 Program) No. 2015AA015304, Shandong Provincial Natural Science Foundation under Grant No. ZR2015FM001, and the Natural Science Foundation of China (NSFC) under Grant No. 61602274.

References

Openflow switch specification, https://www.opennetworking.org/sdn-resources/openflow.

A. unknown, "openflow switch specification," version. pages 1–42, 2011.

Bgp reports,http://bgp.potaroo.net. 2016.

T. Benson, A. Anand, A. Akella, and M. Zhang. Microte: Fine grained traffic engineering for data centers. In *Proceedings of the Seventh COnference on emerging Networking EXperiments and Technologies*, page 8. ACM, 2011.

M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. Gude, N. McKeown, and S. Shenker. Rethinking enterprise network control. *IEEE/ACM Transactions on Networking (ToN)*, 17(4):1270–1283, 2009.

A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee. Devoflow: scaling flow management for high-performance networks. *ACM SIGCOMM Computer Communication Review*, 41(4): 254–265, 2011.

Q. Dong, S. Banerjee, J. Wang, and D. Agrawal. Wire speed packet classification without tcams: a few more registers (and a bit of logic) are enough. In *ACM SIGMETRICS Performance Evaluation Review*, volume 35, pages 253–264. ACM, 2007.

N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker. Nox: towards an operating system for networks. *ACM SIGCOMM Computer Communication Review*, 38(3):105–110, 2008.

Q. Guo, X. Guo, Y. Bai, and E. Ipek. A resistive tcam accelerator for data-intensive computing. In *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 339–350. ACM, 2011.

D. Y. Huang, K. Yocum, and A. C. Snoeren. High-fidelity switch models for software-defined network emulation. In *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, pages 43–48. ACM, 2013.

H. Huang, S. Guo, P. Li, W. Liang, and A. Y. Zomaya. Cost minimization for rule caching in software defined networking. *IEEE Transactions on Parallel and Distributed Systems*, 27(4):1007–1016, 2016a.

H. Huang, S. Guo, J. Wu, and J. Li. Green datapath for tcam-based software-defined networks. *IEEE Communications Magazine*, 54(11):194–201, 2016b.

L.-Y. Huang et al. Reram-based 4t2r nonvolatile tcam with 7x nvm-stress reduction, and 4x improvement in speed-wordlength-capacity for normally-off instant-on filter-based search engines used in big-data processing. In *2014 Symposium on VLSI Circuits Digest of Technical Papers*, pages 1–2. IEEE, 2014.

A. Ishimori, F. Farias, E. Cerqueira, and A. Abelém. Control of multiple packet schedulers for improving qos on openflow/sdn networking. In *Software Defined Networks (EWSDN), 2013 Second European Workshop on*, pages 81–86. IEEE, 2013.

S. Jiang and X. Zhang. Lirs: an efficient low inter-reference recency set replacement policy to improve buffer cache performance. *ACM SIGMETRICS Performance Evaluation Review*, 30(1):31–42, 2002.

X. Jin et al. Dynamic scheduling of network updates. In *ACM SIGCOMM Computer Communication Review*, volume 44, pages 539–550. ACM, 2014.

P. Junsangsri, F. Lombardi, and J. Han. A memristor-based tcam (ternary content addressable memory) cell. In *Proceedings of the 2014 IEEE/ACM International Symposium on Nanoscale Architectures*, pages 1–6. ACM, 2014.

N. Kang, Z. Liu, J. Rexford, and D. Walker. Optimizing the one big switch abstraction in software-defined networks. In *Proceedings of the ninth ACM conference on Emerging networking experiments and technologies*, pages 13–24. ACM, 2013.

K. Kannan and S. Banerjee. Compact tcam: Flow entry compaction in tcam for power aware sdn. In *International Conference on Distributed Computing and Networking*, pages 439–444. Springer, 2013.

N. Katta, O. Alipourfard, J. Rexford, and D. Walker. Infinite cacheflow in software-defined networks. In *Proceedings of the third workshop on Hot topics in software defined networking*, pages 175–180. ACM, 2014.

N. Katta, O. Alipourfard, J. Rexford, and D. Walker. Cacheflow: Dependency-aware rule-caching for software-defined networks. In *Proc. ACM Symposium on SDN Research (SOSR)*, 2016.

U. C. Kozat, G. Liang, and K. Kokten. On diagnosis of forwarding plane via static forwarding rules in software defined networks. In *INFOCOM, 2014 Proceedings IEEE*, pages 1716–1724. IEEE, 2014.

A. X. Liu, C. R. Meiners, and E. Torng. Tcam razor: A systematic approach towards minimizing packet classifiers in tcams. *IEEE/ACM Transactions on Networking*, 18(2):266–275, 2007.

N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. Openflow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, 38(2):69–74, 2008.

G. Rétvári, J. Tapolcai, A. Kőrösi, A. Majdán, and Z. Heszberger. Compressing ip forwarding tables: towards entropy bounds and beyond. In *ACM SIGCOMM Computer Communication Review*, pages 111–122. ACM, 2013.

C. Rotsos, N. Sarrar, S. Uhlig, R. Sherwood, and A. W. Moore. Oflops: An open framework for openflow switch evaluation. In *International Conference on Passive and Active Network Measurement*, pages 85–95. Springer, 2012.

S. Sharma, D. Staessens, D. Colle, M. Pickavet, and P. Demeester. Fast failure recovery for in-band openflow networks. In *Design of reliable communication networks (drcn), 2013 9th international conference on the*, pages 52–59. IEEE, 2013.

R. Shen, X. Li, and H. Li. A hybrid tcam+ sram scheme for multi-match packet classification. In *2012 13th International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT)*, pages 685–690. IEEE, 2012.

J.-P. Sheu and Y.-C. Chuo. Wildcard rules caching and cache replacement algorithms in software-defined networking. *IEEE Transactions on Network and Service Management*, 13(1):19–29, 2016.

B. Stephens, A. Cox, W. Felter, C. Dixon, and J. Carter. Past: Scalable ethernet for data centers. In *Proceedings of the 8th international conference on Emerging networking experiments and technologies*, pages 49–60. ACM, 2012.

D. E. Taylor and J. S. Turner. Classbench: A packet classification benchmark. *IEEE/ACM Transactions on Networking*, 15(3):499–511, 2007.

H.-J. Tsai, K.-H. Yang, Y.-C. Peng, C.-C. Lin, Y.-H. Tsao, M.-F. Chang, and T.-F. Chen. Energy-efficient non-volatile tcam search engine design using priority-decision in memory technology for dpi. In *Proceedings of the 52nd Annual Design Automation Conference*, page 100. ACM, 2015.

Z. Ullah, K. Ilgon, and S. Baeg. Hybrid partitioned sram-based ternary content addressable memory. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 59(12):2969–2979, 2012.

M. Yu, J. Rexford, M. J. Freedman, and J. Wang. Scalable flow-based networking with difane. *ACM SIGCOMM Computer Communication Review*, 40(4):351–362, 2010.

L. Zheng, S. Shin, and S.-M. S. Kang. Memristor-based ternary content addressable memory (mtcam) for data-intensive computing. *Semiconductor Science and Technology*, 29(10):104010, 2014.

L. Zheng, S. Shin, S. Lloyd, M. Gokhale, K. Kim, and S.-M. Kang. Rram-based tcams for pattern search. In *2016 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1382–1385. IEEE, 2016.