Tensor-Based Rule-Space Management System in SDN

Ilora Maity, Student Member, IEEE, Ayan Mondal, Student Member, IEEE,
Sudip Misra, Senior Member, IEEE, and Chittaranjan Mandal, Senior Member, IEEE *

Abstract

This paper presents a tensor-based rule-space management (TERM) system for improving the available capacity of switches in Software Defined Networking (SDN). Limited storage capacity of switches is a key challenge in SDN as the switches use Ternary Content Addressable Memories (TCAMs) having very low capacity. Low rule storage capacity eventually leads to high number of Packet-In messages and control plane overloading. The challenge is to design a dynamic scheme to store a large number of heterogeneous flow-rules in SDN switches and reduce the number of Packet-In messages. To address this problem, we apply the concept of tensor decomposition in order to aggregate flow-rules. In addition, we employ a rule caching mechanism for better throughput. Simulation results show the efficiency of TERM in terms of reduction in the number of Packet-In messages. TERM reduces the Packet-In message count by 57.78% than the flow aggregation approach proposed in the existing literature.

Index terms— SDN, Flow-Rule, TCAM, Tensor Decomposition, Caching

1 Introduction

Software Defined Networking (SDN) provides a global view of the network by separating the control plane from the data plane [1]. This simplifies the task of network administrators and makes SDN suitable for large-scale networks [2]. In particular, researchers are proposing SDN-based architectures for Internet of Things (IoT) applications [3]. These applications involve the management of millions of heterogeneous flows per second [4]. However, the exigency of scalable rule-space in SDN switches is evident [5]. The limited rule-space capacity increases the event of flow-table miss and causes the switch to send Packet-In messages to the controller for the installation of new rules. A Packet-In message contains the header of a captured packet which has no matching entry in the flow-table. The controller formulates new flow-rule based on the contents of the Packet-In message and installs the rule at the corresponding switch. For IoT applications, the number of Packet-In messages can be a bottleneck for the control plane. Therefore, the main objective of the proposed work is to enable the switches to store larger number of rules and decrease the number of Packet-In messages.

1.1 Motivation

State-of-the-art SDN switches store rules in Ternary Content Addressable Memories (TCAMs). TCAM has limited capacity constraint due to its high manufacturing cost [6]. According to the latest OpenFlow protocol version (v1.5.1) [7], each flow-rule contains 45 match fields. Although, most switch manufacturers still use OpenFlow version 1.0, which has 20 match fields, TCAM in these switches can store only up to 8,000 entries [5]. This number reduces when new OpenFlow switches with higher protocol versions are deployed. Additionally, match fields such as the source and destination port numbers are specified as ranges in a typical packet classifier. Multiple flow entries are generated for an equivalent range rule in

*The authors are with the Department of Computer Science and Engineering, Indian Institute of Technology Kharagpur, India (Email: imaity@iitkgp.ac.in; ayanmondal@iitkgp.ac.in; smisra@sit.iitkgp.ernet.in; chitta@iitkgp.ac.in)

a typical packet classifier. In addition, an analysis performed on access control list (ACL) databases of 1998 and 2004 reports that the volume of range rules increased from 1.3% to 13.3% [8]. On the other hand, a large number of applications today provide services to billions of users and this number is predicted to increase rapidly in the future [9]. For example, to serve the users of city similar to New South Wales, an application should be capable of handling 70 million flows per second [4]. This high volume of flows causes rapid and frequent updates in SDN switches, while generating high number of Packet-In messages [10]. Therefore, there is a need for rule-space management in large-scale networks in order to store this high volume of flow-rules.

The existing literature considers three basic approaches — *flow-table aggregation* [11], *flow-rule partitioning* [12] and flow aggregation [13] for handling the capacity constraint of SDN switches. However, the lack of uniformity in handling flow-rules, which are generated from heterogeneous IoT applications, is evident in these approaches. Additionally, these approaches are not flexible enough to provide rule-space and reduce table miss according to the network demand. In this paper, we propose a scheme, named TERM, to increase the scalability of SDN switches and reduce the Packet-In message count.

1.2 Contribution

The primary contributions of this work are listed below.

1. We propose a scheme which is capable of aggregating heterogeneous flow-rules having no common prefix.

2. We envision a tensor-based algorithm to compress rules in each switch.

3. Extensive simulation results demonstrate that TERM significantly reduces the average number of Packet-In messages and increases free space for storing rules.

1.3 Paper Organization

The remainder of this paper is organized as follows. Section 2 discusses the relevant research work. In Section 3, we describe TERM in detail. Section 4 presents the experimental results and comparative studies with other existing approaches. Finally, we conclude the work in Section 5.

2 Related Work

In this section, we review prior works related to flow-table aggregation, flow-rule partitioning and flow aggregation. We also review the existing literature on tensor and singular value decomposition (SVD) used in our work.

Earlier, table aggregation approaches considered only prefix entries, where do not care (∗)s do not appear in the beginning of the ternary strings. Applegate *et al.* [14] proposed a prefix-based minimization technique for access control lists (ACLs), which have entries similar to TCAMs. Meiners *et al.* [11] proposed bit weaving, which partitions the total rule-set and permutes the bit positions for each of the partitions to transform all non-prefix entries into prefix entries. Finally, these transformed partitions are merged together, after which each entry is re-permuted to their original bit order. However, one of the major limitations of bit weaving is high computation time for larger partition size. This is even worse in networks where data changes frequently, because bit weaving recomputes the whole rule-set for each rule update.

Other related works concerns the approach of partitioning the flow-rules. Kanizo *et al.* [12] presented a decomposition technique, which partitions a flow-table into sub-tables and distributes the sub-tables across the network. They proposed two techniques — Pivot Bit Decomposition (PBD) and Cut-Based Decomposition (CBD). PBD decomposes the table into sub-tables by selecting a pivot bit/column. However, PBD increases the total number of rules, because two separate rules are generated for each do not care (*) pivot bits. On the other hand, CBD represents the set of rules by a dependency graph. Moshref

et al. [15] proposed a virtual Cloud Rule Information Base (vCRIB), which partitions the overlapping rules by splitting them. Consequently, the overall number of rules increases.

Flow aggregation approaches minimize the total number of flows in order to reduce the number of flow-rules. Kosugiyama *et al.* [13] proposed an approach which considers end-to-end delay as a parameter of flow aggregation. However, the authors considered delay sensitive flows only.

A tensor [16] is a high-dimensional matrix which represents heterogeneous data. Therefore, some of the existing works use tensors for handling large-scale data [17]. Liavas *et al.* [18] proposed a tensor factorization technique for parallel processing of large-scale data. Motivated by this work, we use tensors for representing large flow-tables. We consider a three-order tensor, where the orders denote a single flow-rule, components of a flow-rule (match fields, priority, and actions), and the total number of flow-rules, respectively. Henry *et al.* [19] presented a Singular Value Decomposition (SVD) technique to extract useful data from a matrix. Schifanella *et al.* [20] implemented an extension to the SVD technique, named TUCKER decomposition [21]. Acar and Yener [22] proposed Higher-Order Singular Value Decomposition (HOSVD), which is a generalization of the TUCKER decomposition. HOSVD imposes orthogonality constraint on component matrices. We use a technique, which is similar to HOSVD, to aggregate the flow-rules by extracting core data from a single dimension which represents the number of entries in a flow-table.

Synthesis. Therefore, we infer that there exist a few works for handling the capacity constraint of flow-tables. However, most of these works do not consider dynamic network traffic which is usual for IoT applications. The proposed scheme, TERM, is distinctive in this respect, because it can extract core data from the entire rule-space, irrespective of the original number of rules. The volume of core data does not depend on the original number of rules.

3 TERM: The Proposed Scheme

In this section, we discuss the network model considered for the proposed scheme, TERM. In addition, we describe the proposed tensor-based approach for managing the rule-space of the switches.

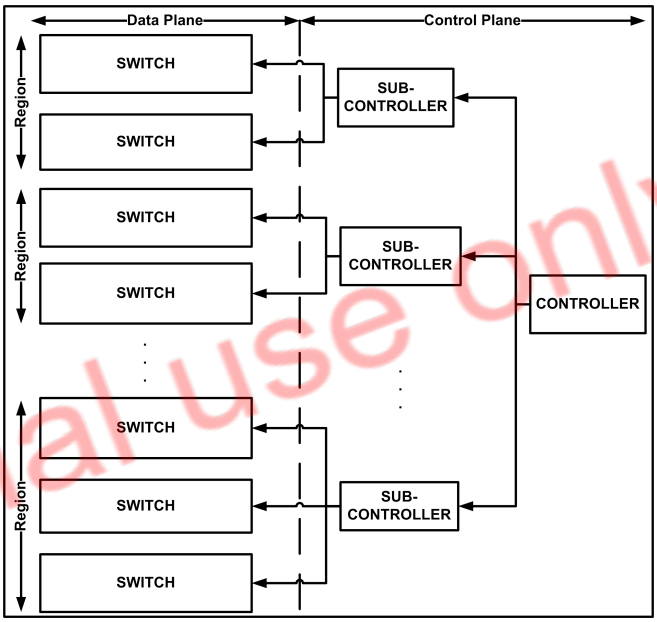

Figure 1: TERM architecture

3.1 Network Model

Figure 1 depicts the network model considered for TERM. The set of switches in the data plane is denoted as $S = \{s_1, s_2, \ldots, s_D\}$. In the control plane, there exist multiple sub-controllers connected with a controller c. The set of sub-controllers is denoted as $C^{sub} = \{c_1^{sub}, c_2^{sub}, \ldots, c_M^{sub}\}$. The sub-controllers are placed using existing controller placement algorithms [4]. All the sub-controllers are connected to c. Each switch s_j is connected to a sub-controller. Therefore, the assignment between switches and sub-controllers is defined as a $D \times M$ binary matrix L. Each element of L is expressed as:

$$l_{ij} = \begin{cases} 1, & \text{if } s_i \text{ is connected to } c_j^{sub}. \\ 0, & \text{otherwise.} \end{cases} \quad (1)$$

***Definition* 1** (Region)**.** *A region r_j is defined as:*

$$r_j = \bigcup_i s_i, \forall l_{ij} = 1 \quad (2)$$

The set of rules in switch s_i at time t is denoted as:

$$R^i(t) = R_c^i(t) \cup R_a^i(t) \cup R_u^i(t), \quad (3)$$

where $R_c^i(t)$ is the set of cached rules, $R_a^i(t)$ is the set of aggregated rules, and $R_u^i(t)$ denotes the set of uncompressed rules in switch s_i at time t.

A switch s_i generates $p_i(t)$ number of Packet-In messages at time t. Packet-In messages are generated whenever incoming packets fail to match with any of the flow-rules in $R^i(t)$.

The objective of this work is to minimize the number of Packet-In messages by maximizing the total number of rules stored in each switch. Mathematically,

$$min \sum_{i=1}^{|S|} p_i(t) \quad (4)$$

subject to

$$|R_c^i(t)| < N_i, \forall s_i \in S \quad (5)$$

$$|R_u^i(t)| < N_i, \forall s_i \in S, \quad (6)$$

where N_i denotes that the TCAM in a SDN switch s_i is capable of storing N_i entries. Equations (5) and (6) express that the number of cached rules and the number of uncompressed rules are less than the storage capacity of the TCAM.

3.2 Tensor-Based Approach for Rule-Space Management

In this section, we describe the proposed scheme, TERM, which includes three modules — rule aggregation, rule reconstruction, and rule caching. Rule aggregation and rule reconstruction procedures of a region r_j are performed by c_j^{sub}. The rule aggregation module compresses the flow-rules in each switch with a tensor-based approach to increase the available capacity of the flow-tables. The rule reconstruction module reconstructs the compressed rules in a switch, whenever an incoming packet fails to match the uncompressed rules. Additionally, each switch has a rule caching module which caches the most frequently used rules. This avoids the reconstruction of rules every time a packet reaches a switch.

Therefore, for an incoming packet, a switch first checks for a rule match in the cached rules, and then the uncompressed flow-rules in each of the flow-tables. If no match is found, it informs the sub-controller that the reconstruction of compressed rules is required. The sub-controller reconstructs the compressed rules and checks for a possible rule match. If a match is found in multiple rules, the higher priority rule is selected. If no match is found even after checking the compressed rules, the packet is redirected to c, which generates the new rule as per existing OpenFlow policy [7].

3.2.1 Rule Aggregation

The rule aggregation module includes three sub-modules — rule restructuring, tensorization, and reduction.

Rule Restructuring Rule restructuring converts the ternary string of each rule into integer value. We consider a 4-bit ternary value for each match field and 4-bit binary value for the action field. Each ternary string of length 2 is transformed to an integer digit based on the transformation rules presented in Table I.

Example 1. *Consider a ternary string of two match fields and one action value* $\{*11*, 10 * 0, 1101\}$*. Therefore, after transformation the resulting integer string will be* $\{35, 82, 97\}$*.*

Tensorization In this work, we use tensor to formalize the flow-tables in SDN switches. We trans-

TABLE I: Integer representation of ternary strings

Ternary String	*Integer Representation*
**	1
*0	2
*1	3
0*	4
1*	5
00	6
01	7
10	8
11	9

form each modified rule-set into a three-order tensor, as shown below:

$$T \in R^{1\times N_f \times N_t}, \tag{7}$$

where N_f denotes the number of fields in a TCAM entry, which includes the priority value, match fields, and action value. N_f depends on the OpenFlow protocol version. N_t denotes the total number of uncompressed rules in the switch.

Algorithm 1 Rule Aggregation Algorithm

INPUT:
1: $T \in R^{1\times N_f \times N_t}$ ▷ `Initial rule tensor`

OUTPUT:
1: $key = \{C \in R^{1\times N_f \times N_r}, U_k\}$ ▷ `Core data set`

PROCEDURE:
1: Compute $T_{(3)}$ from tensor T ;
2: $[USV] \leftarrow SVD(T_{(3)})$; ▷ `Singular value decomposition of mode-3 matrix`
3: Truncate $U_k \in R^{N_t \times N_r}$ from U;
4: $C \leftarrow T \times_3 U_k^T$;
5: $key \leftarrow \{C, U_k\}$;
6: **return** key;

Reduction Algorithm 1 transforms T to a compressed tensor $C \in R^{1\times N_f \times N_r}$, where $N_r < N_t$. N_r is termed as the *reduction factor* (RF). The value of RF at time t is selected as:

$$RF(t) = N_r = N_f + \left\lfloor \frac{(Q_{max} - Q_{current})}{Q_{max}} \times 100 \right\rfloor, \tag{8}$$

where Q_{max} and $Q_{current}$ denote the queue length and the number of packets queued at the switch, respectively. When a switch has high number of queued packets, a low N_r enables the switch to store more number of uncompressed rules.

Therefore, the reduction coefficient is expressed as:

$$q = \frac{N_t - N_r}{N_t} \times 100\% \tag{9}$$

Algorithm 1 reduces the dimensions of the initial rule tensor T and transforms it to the reduced tensor C. In Theorem 1, we discuss that this reduction permits a switch to store more rules, than in the case of a traditional SDN architecture. As we aim to reduce the rule count, we consider the mode-3 unfolded matrix to perform the tensor decomposition method. Mode-3 matrix of tensor T is computed in Line 1 using the procedure of *Tensor Unfolding* or *Matricization* [16]. Figure 2 shows three unfolded matrices of an initial rule tensor $T \in R^{1\times 4\times 8}$, which represents eight flow-rules each with one priority value and two match fields and action value. The corresponding unfolded matrices are $T_{(1)} \in R^{1\times 32}, T_{(2)} \in R^{4\times 8}$, and $T_{(3)} \in R^{8\times 4}$.

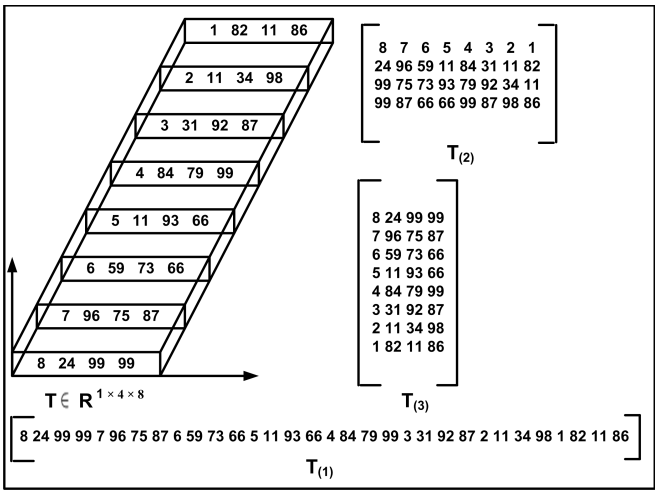

Figure 2: Matricization of initial rule tensor

A tensor element $T(a_1, a_2, ..., a_N)$ for a tensor $T \in R^{I_1\times I_2\times ...\times I_N}$ corresponds to the matrix element $T_{(p)}(a_p, b)$, where

$$b = 1 + \sum_{k=1,k\neq p}^{N} \left((a_k - 1) \prod_{m=1,m\neq p}^{k-1} I_m \right) \quad (10)$$

In Line 2, the unfolded matrix $T_{(3)}$ is decomposed using the singular value decomposition (SVD) technique. SVD factorizes matrix $T_{(3)}$ into the form:

$$T_{(3)} = USV^T, \quad (11)$$

where U and V are the left and right unitary orthogonal matrices, respectively; S is a diagonal matrix, whose elements are singular values of $T_{(3)}$ [19]. Singular values of matrix $T_{(3)}$ are the square roots of the common eigen values of $T_{(3)}T_{(3)}^T$ and $T_{(3)}^T T_{(3)}$. The matrices U and V consist of column vectors, which are transposed eigen vectors of matrices $T_{(3)}T_{(3)}^T$ and $T_{(3)}^T T_{(3)}$, respectively.

The left singular matrix U is truncated in Line 3, which is given by:

$$U_k \in R^{N_t \times N_r} \quad (12)$$

The matrix U_k is needed to be stored for rule reconstruction. We store this matrix U_k in parts in the sub-controllers based on their available memory.

Line 4 generates the compressed tensor C by computing the mode-3 product of tensor T with transpose of matrix U_k, which can be expressed with unfolded matrices as:

$$C = (T \times_3 U_k^T) \Leftrightarrow C_{(3)} = U_k^T \times T_{(3)} \quad (13)$$

Space complexity of Algorithm 1 is $O(N_t^2) + O(N_t(N_r + N_f))$, which decomposes to $O(N_t^2)$ as N_t is greater than both N_f and N_r. The time complexity of performing SVD on the unfolded matrix $T_{(3)}$ in Line 2 is $O(min\{N_t^2 N_f, N_t N_f^2\})$ [23]. The time complexity of computing mode-3 product in Line 3 is $O(N_t N_r N_f)$. Therefore, time complexity of Algorithm 1 is $O(min\{N_t^2 N_f, N_t N_f^2\}) + O(N_t N_r N_f)$. Figure 3 describes the computation of mode-3 product for an order-3 tensor $T \in R^{1\times4\times8}$ multiplied by transpose of truncated orthogonal matrix $U_k \in R^{8\times4}$.

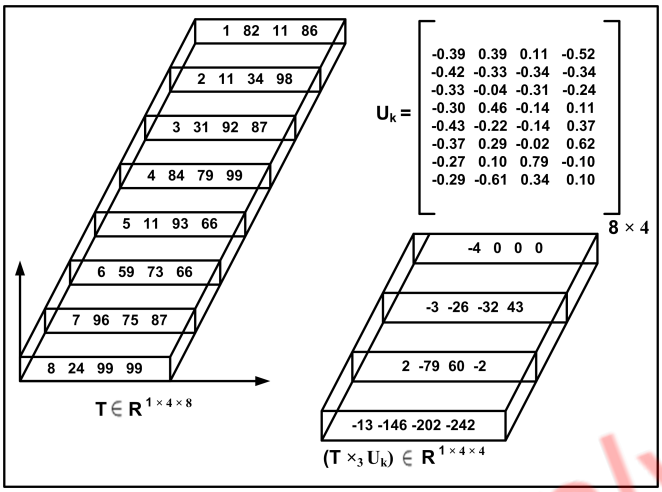

Figure 3: Mode-3 product of the initial rule tensor

The rule aggregation procedure is triggered by the sub-controller if free memory of a switch s_i drops below a certain threshold value th. This limit th is predefined based on nature of the applications. During an aggregation procedure at time t, all the rules in $R_u^i(t)$ are aggregated to form a new set of aggregated rules.

Theorem 1. *The maximum number of rules stored in the TERM SDN architecture is greater than the maximum number of rules stored in a traditional SDN architecture with D OpenFlow switches, where $D \geqslant 1, N > N_f$; N is the storage capacity of each OpenFlow switch in the traditional SDN architecture in terms of the number of entries, and N_f is the number of fields in a TCAM entry.*

Proof. The maximum number of entries stored in a traditional SDN architecture with D switches, each having a TCAM capable of storing N entries, is given by:

$$Max_t = D \times N \quad (14)$$

We denote the maximum number of entries stored in the TERM SDN architecture as:

$$Max_m = D \times \alpha, \quad (15)$$

where α is the storage capacity of each switch in terms of the number of entries of the modified architecture. Therefore, we need to prove that,

$$Max_m > Max_t, \text{ where } D \geqslant 1. \tag{16}$$

Let $T \in R^{1\times N_f \times N_t}$ be the tensor representing rules of a switch with N_t uncompressed entries, where each entry has N_f fields, and $0 < N_t < N$. The corresponding switch contains total $(N - N_t)$ entries comprising of cached entries and the aggregated entries generated from the previous aggregation phase.

The p-mode product of a tensor is the basic flow-rule reduction operation for reducing tensor dimensions. To reduce the dimension of the n^{th} order of a tensor T from I_n to I_p $(I_n > I_p)$, we need to compute n-mode product of tensor T by a truncated left singular vector matrix $U \in R^{I_p \times I_n}$.

The aim of our rule aggregation approach is to reduce the 3^{rd} order of tensor $T \in R^{1\times N_f \times N_t}$ from N_t to N_r, where $N_r < N_t$, to allow storage of larger flow-tables. Therefore, the compressed tensor $C \in R^{1\times N_f \times N_r}$ is expressed as:

$$C = T \times_3 U_3^T, \tag{17}$$

where U_3 is obtained by retaining the left N_r unitary orthogonal vectors from the left singular vector matrix generated from singular value decomposition of mode-3 matrix of tensor T. Figure 3 illustrates the operation of computing compressed tensor C from an initial tensor T. From experimental results, we observe that the minimum value for N_r is N_f for exact reconstruction of flow-rules. Therefore, the maximum percentage of rule reduction for a switch is $\frac{N-N_f}{N} \times 100$ %. N_r can be chosen dynamically depending on the application type. If the application is latency sensitive, then the optimal value of N_r should be chosen, considering the processing time of both rule reduction and reconstruction for approximate rule-set size.

After rule aggregation in each switch, an extra space is available for storing maximum $(N_t - N_f)$ entries. So, L switches can support upto $((N_t - N_f) \times D)$ extra entries. So, Max_m can be expressed as:

$$Max_m = D \times (N + (N_t - N_f)), \tag{18}$$

where the storage capacity of each switch in TERM is $\alpha = (N + (N_t - N_f))$. The term $(N + (N_t - N_f)) > 0$, as $0 < N_f \leqslant 45, L > 0, N_t > 0$, and $N > N_f$ [7]. Hence, the statement of Equation (16) is true.

□

3.2.2 Rule Reconstruction

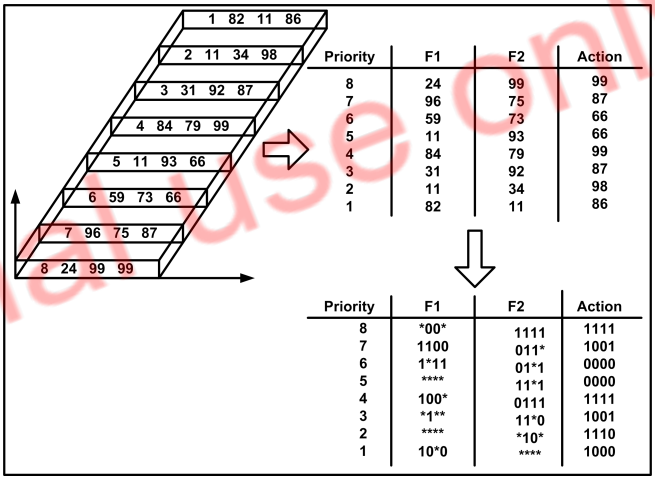

Figure 4: Rule recovery process

The corresponding sub-controller reconstructs the aggregated rules of the switch to verify whether there is a match. Rules in the switch do not change during this process. The reconstructed rules are stored in the sub-controller. After the sub-controller completes the verification process for a possible rule match, it releases the memory used for rule reconstruction.

Change in network policies or topology triggers rule update or addition of new rules. To handle these changes, the aggregated rules of selected switches are reconstructed and then aggregated again after incorporating the changes.

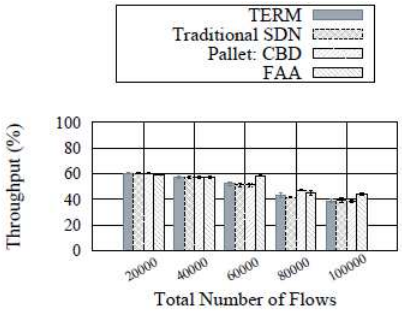

Fig. 5: Average Throughput

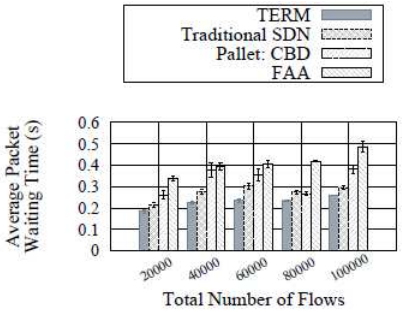

Fig. 6: Average Packet Waiting Time

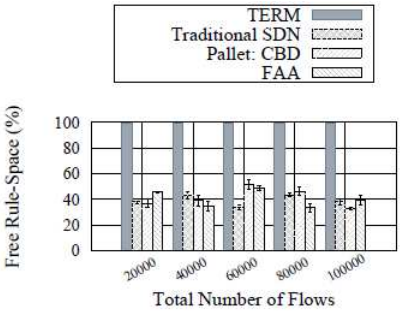

Fig. 7: Average Free Rule-Space

Approximate Rule Tensor Generation For rule reconstruction, initially, we generate an approximate rule tensor $T_A \in R^{1\times N_f \times N_t}$, by computing the mode-3 product of compressed tensor $C \in R^{1\times N_f \times N_r}$ with truncated unitary orthogonal matrix U_k computed using (Equation (12)) and stored. This process is expressed as:

$$T_A = C \times_3 U_k. \tag{19}$$

Equation (19) can also be expressed as:

$$T_{A_{(3)}} = U_k \times C_{(3)}, \tag{20}$$

where $T_{A_{(3)}}$ and $C_{(3)}$ are the mode-3 unfolded matrices of approximate rule and compressed rule tensors, respectively [16]. The space complexity of the rule reconstruction procedure is $O(N_t(N_r + N_f))$. The time complexity of the rule reconstruction procedure is $O(N_t N_r N_f)$.

The absolute error of approximation between initial rule tensor T and approximated rule tensor T_A is measured as:

$$\epsilon = ||T - T_A|| = \sqrt{\sum_{i_1=1}^{1}\sum_{i_2=1}^{N_f}\sum_{i_3=1}^{N_t}(t_{i_1 i_2 i_3} - t_{A_{i_1 i_2 i_3}})^2}, \tag{21}$$

where $||X||$ denotes the *norm* of a tensor X [16]. This error is introduced due to approximation of the floating-point values in the truncated unitary orthogonal matrix U_k. From experimental results, it is observed that $\epsilon = 0$ for $N_r = [N_f, N_t]$.

The size of the matrix U_k depends on RF which we calculate using Equation (8). Hence, the rule reconstruction time is high for a high value of RF due to the computation of mode-3 product in Equation (19).

Rule Recovery After approximate rule tensor is generated, exact rule entries are recovered. Each mode-2 *fiber* [16] of tensor T_A corresponds to one row of flow-table. At this stage, the action value and all the match fields of the flow-table entries are in integer format. Using transformation logic described in Table I, we transform each entry of the flow-table back to ternary string. Figure 4 shows the process of rule recovery.

3.2.3 Rule Caching

Each switch s_i caches the most frequently used rules. Incoming packets, which match with the cached rules, directly follow the actions mentioned in the matched rule. For "cache miss", the packets first search for a match in $R_u^i(t)$. If no match is found, then the corresponding sub-controller reconstructs the aggregated rules and checks for a match.

We used the *least recently used* (LRU) cache algorithm. In the OpenFlow protocol version (v1.5.1) [7], each flow-table entry contains a *counters* field, which is updated when incoming packets are matched with the corresponding flow-rule. Investigation of this field allows us to discard the least recently used rules and select the frequently used ones as the potential caching candidates. The discarded rules are added

to the $R_u^i(t)$ set. If a rule in the aggregated rule-set $R_a^i(t)$ qualifies as a potential caching candidate, then that rule is added to set $R_c^i(t)$ with a flag indicating that the rule is also available in the aggregated rule-set $R_a^i(t)$. Therefore, when the rule is no longer needed to be cached, it can be simply deleted from set $R_c^i(t)$ without adding it to set $R_u^i(t)$.

4 Performance Evaluation

In this Section, we evaluate the efficiency of TERM, while comparing with traditional OpenFlow-based approach, flow-table partitioning approach — *Pallet* [12], and flow aggregation approach (FAA) [13]. We implement all the algorithms in MATLAB and consider Sprint topology [24]. We generate random flow-table entries, each with a priority value, a counter value, 45 match fields, and an action value. The performance of TERM is evaluated based on *throughput*, *average packet waiting time*, *free rule-space*, *Packet-In message count*, and *rule aggregation and reconstruction time*.

TABLE II: Simulation parameters

Parameter	*Value*
Network topology	Sprint [24]
Simulation area	3563.90 km × 1655.20 km [24]
Total number of flows	[20000, 100000]
Number of switches	11
Switch capacity	8000 flow-rules [5]
Packet arrival rate per switch	0.02 mpps [10]
Packet processing rate per switch	0.03 mpps [10]
Rule matching time	20 μs [25]
Transmission delay	5 μs per kilometer [26]
Average queue size per switch	0.07 million packets [10]

4.1 Simulation Parameters

The simulation parameters are depicted in Table II. The total number of switches is 11. The maximum number of flow-entries stored in a switch is 8000 [5]. The average queue size per switch is set as 0.07 million packets [10]. In addition, the rule matching time is fixed to 20 μs [25]. We consider 5 μs transmission delay per kilometer distance [26].

4.2 Result and Discussion

4.2.1 Throughput

Throughput is measured as the percentage of packets processed per unit time. Figure 5 shows the average throughput when the total number of flows is varied between 20000 and 100000. The average packet arrival rate and packet processing rate per switch are 0.02 mpps and 0.03 mpps, respectively. From the simulation, we observe that the average throughput for TERM is almost similar to Pallet, traditional SDN approach, and FAA.

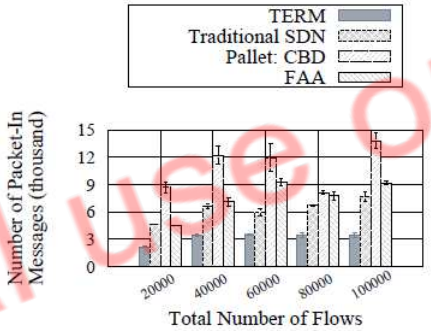

Figure 8: Average Number of Packet-In Messages

4.2.2 Average Packet Waiting Time

Figure 6 depicts the average packet waiting time for TERM, traditional SDN, Pallet, and FAA. The average packet waiting time of TERM is 14.81%, 30.30%, and 43.90% less than traditional SDN, Pallet, and FAA, respectively. Therefore, we yield that the average packet waiting time is short in TERM, as the most frequently used rules are cached in each switch.

4.2.3 Free Rule-Space

The amount of free rule-space is the percentage of total rule-space available for storing new flow-rules. As shown in Figure 7, the average free rule-space is significantly higher in TERM, as each rule aggregation procedure aggregates the existing rules and releases rule-space.

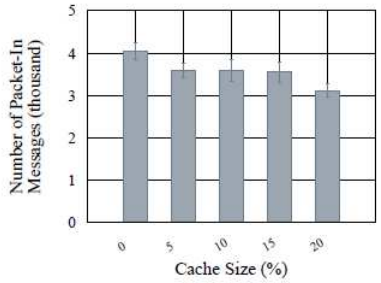

Fig. 9: Effect of Cache Size on Packet-In Message Count

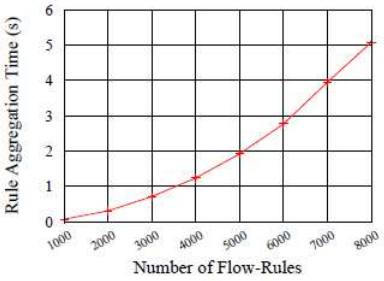

Fig. 10: Rule Aggregation Time

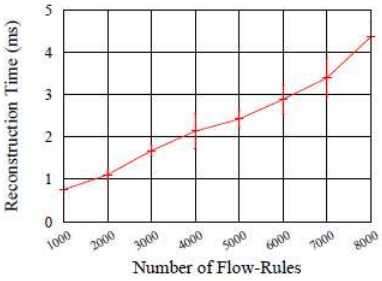

Fig. 11: Rule Reconstruction Time

4.2.4 Packet-In Message Count

Figure 8 shows the average number of Packet-In messages generated from each switch in the network. The cached rule-space size is fixed to 10% of the total rule-space. The average number of Packet-In messages is 49.45%, 70.83%, and 57.78% less than traditional SDN, Pallet, and FAA, respectively.

Figure 9 depicts the average number of Packet-In messages generated from each switch for different cache size. The total number of flows is 10000. As shown in Figure 9, the number of Packet-In messages for 20% cache size is 22.96% less than that for no cache. Therefore, we yield that caching reduces the Packet-In message count. In addition, we synthesize that after a specific size of $R_c^i(t)$, the Packet-In message count decreases as most of the packets are matched in $R_c^i(t)$.

4.2.5 Rule Aggregation and Reconstruction Time

The rule aggregation time is the time required to compress flow-rules of a switch into lesser number of TCAM entries. Similarly, the rule reconstruction time is the time needed to transform the aggregated TCAM entries into actual flow-rules. Figure 10 and Figure 11 depict the average rule aggregation and reconstruction time of a switch, respectively. For 8000 flow-rules, rule reconstruction phase takes 16.25% lesser time than the rule aggregation phase.

5 Conclusion

In this work, we proposed a rule-space management system, TERM, which aims to reduce flow-table miss by increasing the available capacity of switches in SDN. We used tensor decomposition technique to compress heterogeneous flow-rules. We evaluated the performance of the proposed scheme by comparing it with existing approaches. Results indicate enhanced performance in terms of reduced packet waiting time, increased free rule-space, and reduced number of Packet-In messages.

In the future, we aim to extend the proposed scheme by optimizing the rule caching procedure and the placement of flow-rules in SDN switches. The future extension of this work also includes implementation of the proposed scheme in a real testbed.

References

[1] M. Hayes, B. Ng, A. Pekar, and W. K. G. Seah. Scalable Architecture for SDN Traffic Classification. *IEEE Syst. J.*, pages 1–12, 2018. doi: 10.1109/JSYST.2017.2690259.

[2] S. Lange, S. Gebert, T. Zinner, P. Tran-Gia, D. Hock, M. Jarschel, and M. Hoffmann. Heuristic Approaches to the Controller Placement Problem in Large Scale SDN Networks. *IEEE Trans. Netw. Service Manag.*, 12(1):4–17, Mar. 2015.

[3] S. Bera, S. Misra, S. K. Roy, and M. S. Obaidat. Soft-WSN: Software-Defined WSN Management System for IoT Applications. *IEEE Syst. J.*, 12(3):2074–2081, Sep. 2018.

[4] M. T. I. ul Huque, W. Si, G. Jourjon, and V. Gramoli. Large-Scale Dynamic Controller Placement. *IEEE Trans. Netw. Service Manag.*, 14(1):63–76, Mar. 2017.

[5] D. Kreutz, F. M. V. Ramos, P. E. Verssimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig. Software-Defined Networking: A Comprehensive Survey. In *Proc. of the IEEE*, volume 103, pages 14–76, Jan. 2015.

[6] A. Mondal, S. Misra, and I. Maity. Buffer Size Evaluation of OpenFlow Systems in Software-Defined Networks. *IEEE Syst. J.*, pages 1–8, 2018. doi: 10.1109/JSYST.2018.2820745.

[7] OpenFlow Switch Specification(Version 1.5.1): Open Networking Foundation. Mar. 2015.

[8] O. Rottenstreich, R. Cohen, D. Raz, and I. Keslassy. Exact Worst Case TCAM Rule Expansion. *IEEE Trans. Comput.*, 62(6):1127–1140, Jun. 2013.

[9] C. Pereira, A. Pinto, D. Ferreira, and A. Aguiar. Experimental Characterization of Mobile IoT Application Latency. *IEEE Internet Things J.*, 4(4):1082–1094, Aug. 2017.

[10] I. Maity, A. Mondal, S. Misra, and C. Mandal. CURE: Consistent Update With Redundancy Reduction in SDN. *IEEE Trans. Commun.*, 66(9):3974–3981, Sep. 2018.

[11] C. R. Meiners, A. X. Liu, and E. Torng. Bit Weaving: A Non-Prefix Approach to Compressing Packet Classifiers in TCAMs. *IEEE/ACM Trans. Netw.*, 20(2):488–500, Apr. 2012.

[12] Y. Kanizo, D. Hay, and I. Keslassy. Palette: Distributing tables in software-defined networks. In *Proc. of IEEE INFOCOM*, pages 545–549, Apr. 2013.

[13] T. Kosugiyama, K. Tanabe, H. Nakayama, T. Hayashi, and K. Yamaoka. A flow aggregation method based on end-to-end delay in SDN. In *Proc. of IEEE ICC*, pages 1–6, May 2017.

[14] David A. Applegate, Gruia Calinescu, David S. Johnson, Howard Karloff, Katrina Ligett, and Jia Wang. Compressing Rectilinear Pictures and Minimizing Access Control Lists. In *Proc. of ACM-SIAM SODA*, pages 1066–1075, Philadelphia, PA, USA, 2007. Society for Industrial and Applied Mathematics.

[15] Masoud Moshref, Minlan Yu, Abhishek Sharma, and Ramesh Govindan. vCRIB: Virtualized Rule Management in the Cloud. In *Proc. HotCloud*, pages 23–23, 2012.

[16] T. G. Kolda and B. W. Bader. Tensor Decompositions and Applications. *SIAM Review*, 2009.

[17] M. A. Veganzones, J. E. Cohen, R. Cabral Farias, J. Chanussot, and P. Comon. Nonnegative Tensor CP Decomposition of Hyperspectral Data. *IEEE Trans. Geosci. Remote Sens.*, 54(5):2577–2588, May 2016.

[18] A. P. Liavas and N. D. Sidiropoulos. Parallel Algorithms for Constrained Tensor Factorization via Alternating Direction Method of Multipliers. *IEEE Trans. Signal Process.*, 63(20):5450–5463, Oct. 2015.

[19] E.R. Henry and J. Hofrichter. Singular value decomposition: Application to analysis of experimental data. In *Numerical Computer Methods*, volume 210, pages 129 – 192. Academic Press, 1992.

[20] Claudio Schifanella, K. Selçuk Candan, and Maria Luisa Sapino. Multiresolution Tensor Decompositions with Mode Hierarchies. *ACM Trans. Knowl. Discov. Data*, 8(2):10:1–10:38, Jun. 2014.

[21] L. R. Tucker. Some mathematical notes on three-mode factor analysis. *Psychometrika*, 31:279–311, 1966.

[22] E. Acar and B. Yener. Unsupervised Multiway Data Analysis: A Literature Survey. *IEEE Trans. Knowl. Data Eng.*, 21(1):6–20, Jan. 2009.

[23] T. Wu, S. A. Nezam Sarmadi, V. Venkatasubramanian, A. Pothen, and A. Kalyanaraman. Fast SVD Computations for Synchrophasor Algorithms. *Trans. Power Syst.*, 31(2):1651–1652, Mar. 2016.

[24] S. Knight, H. X. Nguyen, N. Falkner, R. Bowden, and M. Roughan. The Internet Topology Zoo. *IEEE J. Sel. Areas Commun.*, 29(9):1765–1775, Oct. 2011.

[25] K. Sood, S. Yu, and Y. Xiang. Performance Analysis of Software-Defined Network Switch Using $M/Geo/1$ Model. *IEEE Commun. Lett.*, 20(12):2522–2525, Dec. 2016.

[26] T. Mizrahi, E. Saat, and Y. Moses. Timed Consistent Network Updates in Software-Defined Networks. *IEEE/ACM Trans. Netw.*, 24(6):3412–3425, Dec. 2016.