Toward migration to SDN: Generating SDN Forwarding Rules by Decision Tree

Sawsan Youssef
Brno University of Technology
Brno, Czech Republic
iyoussef@fit.vutbr.cz

Ondrej Rysavy
Brno University of Technology
Brno, Czech Republic
rysavy@vutbr.cz

***Abstract*—The deployment of Software Defined Network (SDN) switches faces various challenges, one of them is to generate rules to be preloaded in the flow table for performance improvement. Since the flow tables in SDN are implemented by TCAM (ternary content-addressable memory), they have a limited capacity. On the other hand, prepopulating them with the most common rules will reduce the flow setup delay. This paper provides a method for the identification of the most suitable candidate rules by observing the existing traffic and automatic generation of OpenFlow rules that can fit in SDN tables. The rules are extracted from the decision tree build based on the observed NetFlow traffic. The experiments showed that this method can provide a compact set of rules matching most of the network traffic (no less than 98 %).**

***Index Terms*—Decision tree, Software Defined Networks, Data Mining, Netflow, Flow Table, Rule Generation.**

I. Introduction

Software-Defined Networking (SDN) separates network control from the data plane. While this allows for great flexibility, it also involves a performance penalty when packets cannot be processed on an SDN switch due to a missing match rule. When migrating to SDN, we usually require preserving existing network services. The approach may be to analyze the existing configuration and transform it into an SDN program or to monitor existing communication to derive appropriate SDN rules automatically.

Due to the limited capacity of the rule table of SDN switches (1500–3000 rules), it is not always possible to install all the required rules. Unmatched flows are resolved by the SDN controller, which introduces a flow setup delay. In this paper, we present a method that uses data mining techniques to identify the most appropriate rules for SDN switches based on the observation of NetFlow records collected from the network.

A. Contribution

We propose a method to compute a compact set of flow rules that describes the observed traffic. The size of the rule set is important to fit them into the SDN switch and to avoid costly rule lookups in the SDN controller. We use a tree representation of rules, which is convenient for identifying and removing redundant and overlapping rules. We build on the work presented by Wakabayashi et al. [1], where the method of compression of filtering rules in IP networks is presented. CART and C4.5 decision trees represent rules with ”allow” or ”disable” actions. The authors determine the accuracy of the decision tree but do not provide any verification of correctness. Also, only source-destination rule headers without QoS or multiple paths that lead to incorrect forwarding flows are considered. To increase the utilization of the SDN switch and create fewer SDN forwarding rules, we classified the NetFlow traffic according to its exit port (to represent the next hop of routing). More rule headers, such as input port and protocol type are included to reduce the number of incorrect forwarding flows and to consider QoS routing. This method can be applied periodically in dynamic networks. We verify the proposed method’s robustness with several experiments: testing and emulating the SDN framework (mininet) along with different types of datasets.

B. Paper Organization

The organization of this paper as follows: Related work is discussed in section II, followed by the preliminaries and description of the proposed method in section III. We presented the evaluation on several data sets in section IV. The results are discussed in section V followed by a summary in section VI.

II. Related Work

Our work is motivated by the issue of replacing IP routers with SDN devices, which requires transforming the router configuration to SDN programs. For efficiency, compaction of the rule is required to fit in the SDN switch. This section provides an overview of the closest related work in this area.

Due to the resource limitation in SDN networks, the optimization of the number of efficient rules that could be stored in SDN switches should be satisfied. Although the optimizing of the two-dimension routing rules is NP-hard [2], several types of research were applied in the TCAM compression and minimization of SDN forwarding rules. An example is Officer [3], where the SDN network is treated as a black box that should satisfy required end-to-end policy and the optimal resource allocation which relies on distributing the rules for default and alternative paths.

Another line of research suggests using wildcard rules to minimize the rule space. In DomainFlow [4], the network flow is controlled by two sets of rules: one with wildcard and one with exact match rules, which can cause forwarding problems and potential security risks. Minnie [5] reactively compresses the SDN rules by using three aggregation methods (according to the source, destination, and hybrid), then choosing the minimal results of the three algorithms. The drawback is the high complexity and processing and the possibility of improper forwarding of new traffic. The compression technique in [6] relies on the IP mask reduction technique and excludes the rules covered by a parent rule. Also, such an approach was applied in the proposed aggregation of SDN rules in [7]. The approach followed in [8] to solve the rule placement problem is to pre-define the paths that packets must follow inside the network (e.g., shortest path) and then to determine the set of rules and their placement (supposing that the paths are respected). A dependency graph was applied in Pallete [9] to cut the rule table into several tables before distributing them to the OpenFlow switches. Another method, ReWiflow [10] reduced the OpenFlow rules by changing the header fields' order in the flow entries. The header field appears in the flow entries just if the previous header is stored. The existing methods often use only a few rule features (source, destination, input, and output) which may be the limitation when considering, e.g., the multi-path routing or QoS policy.

In our previous work [11], the partial node replacement of the traditional network into SDN based on the NetFlow traffic was discussed. We proposed an algorithm for NetFlow traffic analysis and OpenFlow rules generation (exact match rules). The present work improves the approach by reducing the number of resulted rules.

III. Proposed Method

The method accepts Netflow records of modeled network traffic (Section III-B) and computes a set of compressed SDN rules as a result of several computational steps consisting of creating a decision tree from NetFlow rules (section III-C), generating rule candidates (section III-D) and their further ordering and prioritization (section III-E). Finally, the rules are validated, and redundancy is removed (section III-F).

A. Preliminaries

The routing and firewall policies are implemented in the SDN network as a set of flow rules that control switches. Rules are installed into the OpenFlow tables with respect to their priority. Every rule r is a tuple $r = (Pri, Pat, Pts)$ where: Pri is the rule priority, Pat is the matching pattern, and Pts is a set of output ports. The packet can be forwarded to one of the output ports (unicast), a group of ports (broadcast/multicast), the controller (output port ctrl), or null port (packet is dropped). The action of matching rule with the highest priority is chosen to process the packet. However, when there are two rules with the same priority that match the same packet, the OpenFlow switch can arbitrarily choose one of them [12]. The symbols used in the following description are listed in Table I.

TABLE I
Formal model symbols

Symbol	Description
Rec	Number the data set records
T_r	Tree rules (tree leafs)
S_r	SDN rules of switch S
Acc	Decision tree accuracy
FF	False forwarded flows according to the constructed rules
$Ports$	Number of the output ports of the router
C_f	Coefficient variation of feature f
Fl_f	Set of flow features
BS_r	Generated SDN rules by applying bitmask ports
T_{depth}	Max depth tree

B. Traffic Processing

At this step, the NetFlow traffic collected at the switch is exported to CSV data sets. Then data is processed and flow features Fl_f (sa: source IP, da: destination IP, in: input port, sp: source port, dp: destination port, pr: protocol, out: output port) are extracted by Python scripts from NetFlow data for each record. Note that for the decision tree all feature values are converted to integer values. IP addresses are considered as 32-bit size integers. The protocol names are mapped back to their numbers[1], e.g., "TCP" is presented as 6 and "UDP" as 17.

C. Decision Tree Modeling

Tree based algorithms are used to explore the patterns in large datasets, in this case, we employ them to model the characteristic of observed network traffic. To prevent the decision tree from over-fitting, several methods to perform pruning in the decision tree are applied. We use the post-pruning which is parameterized by the cost complexity parameter ($\boldsymbol{\alpha}$) to control the size of the decision tree and to select the optimal tree size.

As $\boldsymbol{\alpha}$ increases, more subtrees are pruned. The goal is to find a value such that we get the best matching accuracy for rules that fit the OpenFlow table. We implemented a Python script to calculate the range of suitable $\boldsymbol{\alpha}$ values. The script determines the depth of the tree and achievable accuracy (to be evaluated according to the resulted SDN rule).

D. Generating SDN rules

This step takes the root of a decision tree computed in the first step and generates *tree rules* ($\boldsymbol{T_r}$) by enumerating all downward paths to leaves. Every path consists of a set of conditional expressions and the decision assigned to the leaf node. The rules are exported to a CSV file from which the *SDN rules* ($\boldsymbol{S_r}$) are listed using the Algorithm 1 as follows:

- For every resulted tree rule $\boldsymbol{r_i}$, the $\boldsymbol{FeatureSet(r_i)}$ that represents the feature constraints that are contained

[1]https://www.iana.org/assignments/protocol-numbers/protocol-numbers.xhtml

Algorithm 1 Rule Generation

```
Input: $T_r$ - set of the tree rules
Output: $S_r$ - set of the SDN rules

1:  for all $T_{r_i} \in T_r$ do
2:      for all $F_j \in FeatureSet(T_{r_i})$ do
3:          if $F_j$=$sa$ or $F_j$= $da$ or $F_j$= $sp$ or $F_j$= $dp$ then
4:              Compute the range $r_j$ of $F_j$
5:              $(value, mask)$ := Range_GetValueAndMask($r_j$)
6:              $S_{r_i}[F_j] := WildcardMatch(value, mask)$
7:          else
8:              $value$ := GetValue($F_j$)
9:              $S_{r_i}[F_j] := ExactMatch(value)$
10:         end if
11:     end for
12:     Set_Add($S_r$, $S_{r_i}$)
13: end for
```

in the tree rule are extracted. Then the constraints are evaluated to obtain resulting range using interval operations (Line 3). For example, source address constraints $sa \leq 484848$, $sa \leq 4545$, $sa > 323$ yields to range $sa \in [324, 4545]$. When the lower or upper bound is undefined, we use the smallest IP address or largest address that occurs among the training data. In [1], they use different default bounds for IP addresses, namely "255.255.255.255" for the upper and "0.0.0.0" for the lower, which can cause more wild card rules.

- After determine the ranges, the rules will be produced by encoding the ranges using bit masks. The set of IP addresses in a range is set by using Python package $netaddr$, which computes a set of network addresses and their masks. As any match fields of OpenFlow rule may be wild-carded using a bit mask we can directly use the computed network address and mask in the rule for both source and destination address fields match. Moreover, we compute bit masks also for other features, e.g., port and protocol ranges[2].

E. Sorting and Prioritization

Each rule should be prioritized in the generated SDN rule set to avoid conflict when multiple rules may match traffic. Note that the tree path conditions for fields are disjoint; thus, the priority of the rules is related to the fields that are not included in the match. For example, in the decision tree from Fig.1, each node represents a condition phrase related to a particular feature. Since a rule may not cover all fields when matching, it is preferable to prioritize more specific rules over general ones.

For two rules from Table II, the ranges of every feature of rules are as depicted in Table III. The range representation was

[1]https://pypi.org/project/netaddr/

[2]In actual implementation the protocol is not represented using ranges.

TABLE II
EXAMPLE OF THE TREE RULES WITH CLASS 0 AND 4 OF FIG.1

$Tree\_Rule$	Path conditions	out
T_{r_1}	$da \leq 587202688$, $sa \leq 251658496$	0
T_{r_2}	$da > 587202688$	4

TABLE III
RESULTED FEATURES RANGE OF THE RULES IN TABLE (II)

Tree Rule	Features	Ranges
T_{r_1}	sa	[167772417, 251658496]
	da	[335544578, 587202688]
T_{r_2}	da	[587202689, 671088643]

discussed in Section III-B (where the lower IP address of da is the smallest IP address of the destination IP address among the training dataset).

One of the SDN generated rule S_{r_1} based on T_{r_1} is: $priority = 600, dl\_type = 0x800, nw\_src = 10.0.1.0/24, nw\_dst = 20.0.1.2/31, actions = drop$ Where $dl\_type = 0x800$ refers to the data link type to use Ipv4 in the rule. The priority was calculated by the following equation:

$$\text{priority} = \text{default} - 100 * \text{number of empty fields}$$

For example the priority of S_{r_1} is: $1000 - 100 * 4 = 600$. For the T_{r_2}, the SDN resulted rule S_{r_2} is: $priority = 500, dl\_type = 0x800, nw\_dst = 40.0.0.0/30, output = 4$.

F. Validation and Redundancy removing

Redundant rules may appear due to rules that cover all assumed continuous ranges in the decision tree and do not match any traffic from the training streams. We exported the rules that match the original traffic and removed the redundant rules. The correctness of the generated rules is checked by comparing the output field of the record before and after matching with the generated rules.

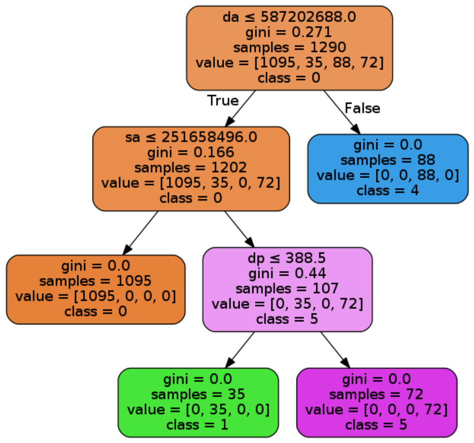

Fig. 1. Decision tree example

TABLE IV
COMPARISON OF SEVERAL ATTRIBUTES OF THE USED DATASETS.

$Data$	Rec	C_{sa}	C_{da}	C_{sp}	C_{dp}	$Ports$
$D2$	1048576	27.43	27.49	77.73	81.19	5
$D_{250,0.001}$	83196	43.75	45.19	102.60	102.65	2
$D_{500,50}$	5454	30.99	29.41	66.67	80.17	4
$D_{1000,0.001}$	5485	40.66	41.40	91.13	84.42	3
$D_{1000,10}$	5485	40.67	41.40	91.13	84.42	3
$D_{1000,10}$	2647	47.16	43.44	100.241	93.56	3
$D_{c,50}$	2647	34.72	31.44	67.54	77.31	3

IV. EXPERIMENTS

For demonstration and evaluation of the method we run two types of experiments:

1) Validation by testing: Several Python scripts for matching the original flows against the generated rule are used. We run the experiments on public existing dataset [3]:
The traffic was generated by using DOcker-based fRamework fOr gaTHering nEtflow (DOROTE) traffic. Only the dataset with input packet sampling (the method to place less load on the network devices by collect subset of the packets instead of every packet) was used in our validation. The datasets contain also attack flows which nevertheless has not negative effect the method. However, in this work we assume that all traffic is regular and thus we learn rules from both normal and attack flows. Generating rules that can deny attack flows for labeled input is left for future work.
We have tested the method with datasets of various sampling rates and attacks included. The datasets are referred by names $D_{x,y}$ where: x refers to the sampling rate and y refers to the attack percentage in the dataset. Also we have evaluated the method using another public dataset (denoted as $D2$ [4]) that contains million NetFlow records.
Table IV overviews attributes that characterizes the datasets. Column Rec denotes the total number of NetFlow records. The coefficient of variation CV of features is used to compare the variation between two different data sets.
In the experiments, the SDN rules are generated initially without pruning. If the number of rules exceeds the switch capacity (3000) the port bitmask rules will be generated. If this does not help the pruning is applied and adjusted by α parameter.
2) Evaluation in Mininet: We evaluated the proposed framework in SDN network by Mininet emulation tool. First, we converted the traditional network BSO Network 2011 topology obtained from Topology Zoo [5] to SDN network.
The network consists of 14 nodes and 18 links. The default forwarding rules are installed in the SDN switches according to the shortest path. Each node is represented by OVS-switch that is connected to a host and generates traffic flows in the network. Prepared Pcap files were collected in real network from several sources (real-time captured in public Pcap library: Netresec [6]. The Pcap files traffic are variant: File transfer, video stream, DNS, web surfing and even intrusion traffic. TcpReplay was used to resend the Pcap files into the network. Second, the observed traffic at SDN switch is exported to an external NetFlow server. Such dataset is processed by the proposed method to generate the new rules. For validation, the generated forwarding rules are reinstalled at the SDN switch (after deleting its original configurations). Finally, original traffic is replayed again in the network and matched against the new configurations.
In our simulation testbed in SDN, two datasets were tested: Dataset D_A which contains 1758 distinct flows of NETREC (gathered in mininet during 3 days). Another dataset D_B was gathered during half an hour (404 distinct rules: The same transport protocols of NETREC dataset, same IP addresses range, 20 sessions for every pair of hosts).

[3]https://open.scayle.es/dataset/netflow-data-with-ng-for-test
[4]https://zenodo.org/record/4106738#.YKPWM6gzY2x
[5]http://www.topology-zoo.org/dataset.html

V. RESULTS

The comparison of the results of the public data sets is shown in Tables (VII,VI). Validation test in mininet emulation is depicted in table (V). We discovered that the proposed algorithms even when α = 0 minimizes the generated rules with small number of false forwarding rules (Table VII). The coefficient variations of the features (Table IV) affects the number of features that are included in the tree nodes. The small number of the coefficient variations of the sources and the destinations of numerous traffic D_2 improves the rules reduction. The algorithm is calculated for best predicted α =0.004 and 0.001 in table VI, and still the final rules are fitted in SDN switch (less than 3000 rules).

The resulted forwarding rules are matched against the training data to measure the the flows that are incorrectly forwarded.It is shown in the results that FF is minimized when the pruning is low. The final generated rules can be calculated as a union of the resulted rules set based on the mining method and the false-forwarding rules set (which will be exact match rules assigned with higher priority) and expressed as:

$$Ef_r \cup FF$$

Where Ef_r is the resulted rules (without redundant).

To present the efficiency of generating bitmask rules, the comparison of the generated rules of the dataset (D_B) is shown in (Fig. 2) and the violated flows in Fig. 3. The OVS bitwise rules efficiently reducing the number of rules with negligible difference number of incorrect flows when it is not used. For the dataset ($D_{250,50}$: sampling 250/attack 50%), the

[6]https://www.netresec.com/

TABLE V
RESULTS OF ALGORITHM IN MININET ENVIRONMENT.

$DataSet$	α	Ef_r	FF
D_A	0	59	21
	0.01	11	193
D_B	0	57	5
	0.01	39	13

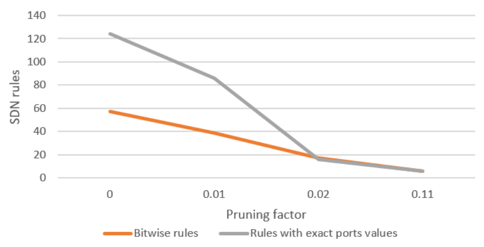

Fig. 2. Rules comparison in D_B dataset (with and without Bitwise ports)

number of generated rules without bitmask is 4358 which is minimized into only 26 rules (with bitmask), without changing of the violated flows (5 violated flows).

A. Comparison with previous work

Most of the previous work related to the SDN rules compression didn't evaluate excessive forwarding rules. For example, the number of the dataset rules evaluated in [1] is 1555 rules, and is minimized into 124 rules with an accuracy 92%). In Minnie [5] the SDN switch rules limits was 750 rules that are re-actively grouped with negligible loss rate. 1636 rules of Stanford dataset are compressed by Reviflow [10] (reactively) by covering 85% of the original flows.

In this work, several types of datasets were tested (ranging from several hundreds of records to millions of records (as in D_2 dataset which is minimized into 143 effective rules and 270 incorrect forwarding rules (less than 0.02 %)). The rules that express the resulted violated flows are integrated with the wildcard rules To improve the accuracy.

VI. SUMMARY

This paper described and evaluated a new SDN forwarding rule generation method based on the decision tree classification

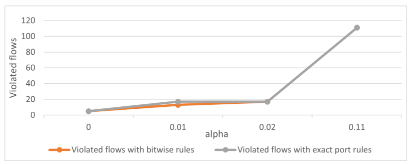

Fig. 3. Violated flows with/without Bitwise in D_B

TABLE VI
RESULTS OF ALGORITHM ON D_2 DATABASE.

α	T_r	BS_r	EF_r	FF
0.001	9	6672	143	270
0.004	4	2174	139	1890

TABLE VII
RESULTS WITHOUT PRUNING (α =0).

dataset	T_r	Acc	S_r	FF	Ef_r
$D_{250,0.001}$	3	0.999	3	1	3
$D_{500,50}$	7	0.991	47	9	37
$D_{500,10}$	5	0.991	70	3	65
$D_{1000,0.001}$	4	0.999	70	3	7
$D_{1000,10}$	6	0.998	43	3	3
$D_{1000,50}$	5	0.989	38	20	7

of monitored NetFlow traffic. The experimental results showed that it is possible to generate rules that fit into the SDN flow table and still provide high accuracy. Validation of rules in the SDN environment was tested on a real dataset in Mininet. Further, future work is to evaluate the proposed method in different topological models with multipath routing.

ACKNOWLEDGMENT

This research was supported by project *Smart Information Technology for Resilient Society* financed by the Internal Grant Agency of Brno University of Technology.

REFERENCES

[1] K. Wakabayashi, D. Kotani and Y. Okabe, "Traffic-aware Access Control List Reconstruction," 2020 International Conference on Information Networking (ICOIN), Barcelona, Spain, 2020, pp. 616-621, doi: 10.1109/ICOIN48656.2020.9016512.
[2] F.Giroire, F.Havet and J.Moulierac. Compressing two-dimensional routing tables with order. INOC (International Network Optimization Conference), May 2015, Varsovie, Poland. pp.351-358.
[3] X. Nguyen, D. Saucez, C. Barakat and T. Turletti, "OFFICER: A general optimization framework for OpenFlow rule allocation and endpoint policy enforcement," *2015 IEEE Conference on Computer Communications (INFOCOM), Kowloon*, 2015, pp. 478-486, doi: 10.1109/INFOCOM.2015.7218414.
[4] Y. Nakagawa, K.Hyoudou, C.Lee, S. Kobayashi, O. Shiraki, T.Shimizu, " Domainflow: Practical flow management method using multiple flow tables in commodity switches." *In Proceedings of the Ninth ACM Conference on Emerging Networking Experiments and Technologies* , Santa Barbara, CA, USA, 9–12 , December 2013.
[5] M. Rifai et al., "Too Many SDN Rules? Compress Them with MINNIE," *IEEE Global Communications Conference (GLOBECOM)*, San Diego, CA, 2015, pp. 1-7, doi: 10.1109/GLOCOM.2015.7417661.
[6] H. Liu, "Routing table compaction in ternary CAM," *in IEEE Micro*, vol. 22, no. 1, pp. 58-64, Jan.-Feb. 2002, doi: 10.1109/40.988690.
[7] Yu, Curtis and Lumezanu, Cristian and Madhyastha, Harsha V. and Jiang, Guofei", "Characterizing Rule Compression Mechanisms in Software- De
ned Networks","Passive and Active Measurement conference, Springer International Publishing, 2016, 302-315
[8] X. Nguyen, D. Saucez, C. Barakat and T. Turletti, "Rules Placement Problem in OpenFlow Networks: A Survey," *in IEEE Communications Surveys and Tutorials*, vol. 18, no. 2, pp. 1273-1286, Second quarter 2016.
[9] Y.Kanizo,D.Hay and I. Keslassy: Palette: Distributing tables in software-defined networks. IEEE Infocom Mini-conference, Apr 2013.
[10] S.Sajad and G.Yashar: ReWiFlow: Restricted Wildcard OpenFlow Rules. In ACM SIGCOMM Computer Communication Review. Volume 45, Number 5, October 2015.
[11] S. Youssef and O.Rysavy ," Proposed Method for Partial Node Replacement by Software Defined Network", Federated Conference on Computer Science and Information Systems, ACSIS, Vol. 22, pages 11–14 (2020)
[12] ONF, OpenFlow Switch Specification Version 1.5.1(Protocol version 0x06), March 26, 2015.