# R2-D2: Filter Rule set Decomposition and Distribution in Software Defined Networks

Ahmad Abboud[†*], Rémi Garcia[‡*], Abdelkader Lahmadi[*], Michaël Rusinowitch[*], Adel Bouhoula[§†]
[*] Université de Lorraine, CNRS, Inria, Loria, F-54000 Nancy, France, {firstname.lastname}@inria.fr
[†] NUMERYX, France, a.abboud@numeryx.fr, a.bouhoula@numeryx.fr
[‡] IPB Enseirb-Matmeca, France, rgarcia003@enseirb-matmeca.fr
[§] College of Graduate Studies, Arabian Gulf University, P.O. Box 26671, Kingdom of Bahrain, a.bouhoula@agu.edu.bh

***Abstract*—Software Defined Networks administrators can specify and smoothly deploy abstract network-wide policies. The rule sets of these policies are deployed in the forwarding tables of the available switches. In this paper, we propose a technique, named R2-D2, for decomposing and distributing a rule set on network switches of limited flow tables size, while preserving the network policy semantics. Through experiments on several rule sets with single dimension, we evaluate and analyse the performance of our rule decomposition techniques. Our results show that our technique is efficient in practice compared to existing techniques.**

## I. Introduction

In software-defined networks (SDN), the filtering requirements for critical applications often vary according to flow changes and security policies. SDN address this issue with a flexible software abstraction, allowing simultaneous and convenient modification and implementation of a network policy on flow-based switches. This single-point deployment approach constitutes a key feature for complex network management operations.

The growing number of attacks coming from diverse sources continually increasing the number of entries in access-control lists (ACL). To avoid relying on large and expensive memory capacities in network switches, a complementary approach to rule compression [1], [2], [3] would be to use multiple smaller switch tables to enforce in the network the access-control policies. It paves the way towards distributed access-control policies, which have been the topic of many previous studies [4], [5], [6]. However, most of them have a large rules replication [4], [5] or even they modify the header of the packet [6] to avoid the matching of a rule by a packet in the next switch.

In this work, we tackle the problem of rule-placement of ACLs, while focusing on three major challenges:

- Simplicity and efficiency of rule set decomposition to facilitate security policy deployment and updating;
- Decomposition over diverse network topologies;
- Decomposition of complex network policies.

In this paper we introduce a new technique, named R2-D2, to decompose and distribute filtering rule sets over a given network topology. Our approach is to design decomposition schemes for both simple and complex policies relying on a single dimension.

Like Palette [5] and OBS [4], we rely on rule dependency relations, but we generate less forward rules by taking the action field into consideration. Indeed, a packet is allowed to be matched by rules with two prefixes from successive switches, if their resulting action does not violate the initial policy semantics.

In the following sections, we introduce efficient decomposition and distribution algorithms for network policy management as well as the decomposition possibilities related to different topologies. In Section II, we state the general principles and distribution constraints in this problem. In Section III, we perform a forward table decomposition using the commonly used Longest Prefix Matching (*LPM*) priority strategy in OpenFlow [7] wildcard matching. Section IV details the distribution of rule tables with respect to relative positions of neighbour devices. Section V presents some benchmarks. We present our conclusions in Section VI.

## II. Problem statement

### A. Definitions

In this paper, we are concerned by decomposing and distributing a set of filtering rules along switches of network $N$ in order to satisfy a given SDN policy while meeting the capacity limitation of each switch.

We first consider the case where rules are applied with a Longest Prefix Matching (LPM) strategy. According to this strategy, one packet can match multiple rules, but only the one with the most specific matching prefix (i.e., the longest prefix) will be selected. In the example of Table I, if a switch receives a packet with 1001 as address, and using a prioritized list strategy, the first rule's action $A_1$ should be applied. However, with an $LPM$ strategy, the packet matches both Rules 1 and 2, but only action $A_2$ will be applied since Rule 2 covers the address field with a longer prefix.

| Rule | Address field | | | | *Action* |
|---|---|---|---|---|---|
| 1 | 1 | 0 | * | * | $A_1$ |
| 2 | 1 | 0 | 0 | * | $A_2$ |

TABLE I: Example of a rule set in a switch table.

978-3-903176-31-7 

Here, we consider that filtering rules have two possible actions: "Forward" or "Deny". In the following, the prefix of a rule $r$ is denoted by $Pref(r)$ and its action by $Action(r)$. If $Pref(r_1)$ matches $Pref(r_2)$ and has a shorter bit-prefix than $Pref(r_2)$, like Rule 1 with Rule 2 in Table I, then we say that $r_1 > r_2$ or $r_1$ overlaps $r_2$. In the example of Table I, it is also true that $Pref(r_1)$ is the next longest matching prefix i.e there is no $r$ such that $r_1 > r > r_2$. We express this by $r_1 >: r_2$.

### B. Distribution and decomposition requirements

To ensure that distributing the rules along different switches does not change the initial policy compared to a single-switch placement, we must preserve its overall semantics, i.e., *the action applied on a matched packet in a single switch with the initial policy rule set should be the same action in a chain of switches with the distributed policy rule set.*

Let $R$ be the initially given rule set of a policy. Let $R_1$ and $R_2$ be two subsets of $R$ located in different switches along a path, with $R_2$ being located in a switch after $R_1$'s one as illustrated in Fig. 1.

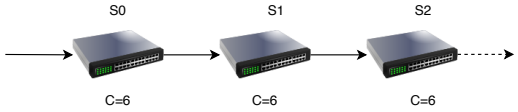



Fig. 1: Flow table capacities of switches along a single path.

When applying the $LPM$ strategy in a switch, a packet must be processed by the most specific matching rule. Thus, **if $r_2 \in R_2$, there must not exist any $r_1 \in R_1$ such as $r_1 > r_2$**. To enforce this precedence property, when we place a rule $r$ in a switch, we must also place in the same switch any rule $r'$ such as $r > r'$.

When a packet has been accepted by a switch containing rule set $R_1$ and enters a switch containing rule set $R_2$, if $r_1$ and $r_2$ are matching rules in $R_1$ and $R_2$ respectively, and $r_2 > r_1$ then $r_2$ should not block it. Thus, we must prevent blocking a packet when there is an action conflict. Given two rules $r_1 \in R_1$ and $r_2 \in R_2$, we have an action conflict iff $r_2 >: r_1$, $Action(r_1) =$ "$Forward$" and $Action(r_2) =$ "$Deny$".

When decomposing $R$ into subsets to be stored in the different switches and adding forward rules to solve action conflicts, the decomposition problem is formulated as a Bin Packing problem with fragmentable items [8] where each fragmentation induces a cost.

## III. Decomposition over a path

In this section, we present our decomposition algorithm over a single path using LPM strategy with a single filtering field. We use a binary tree to represent the rules ordered by their prefix specificity. We study the single field case, so each rule contains one filtering field and there is at most one rule associated to a tree node. The sequence of edge labels taken from the root to a node represents the bit-prefix of the rule linked to this node.

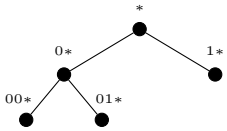



Fig. 2: Compact representation of rules prefixes in a binary tree.

As shown in Fig. 2, the pattern $00*$ of a rule $r$ is at distance 2 from the root, reachable through the leftmost branch of the tree. The complete prefix with wildcards of a node $n$ represents a rule pattern. Once the rules are distributed among the switches, if in any switch no matching rule is found for an incoming packet, the packet will be forwarded to the next switch using a default rule.

### A. Forward rules generation

In order to respect both the initial $R$ and $LPM$ semantics, forward rules need to be generated.

Given a set of rules $R$, two rule prefixes $p_1 = w0*, p_2 = w1*$ can be merged to obtain a forward rule prefix $p' = w*$. This operation can be iterated.

Fig. 3 illustrates a rule tree with merging possibilities. This tree contains 4 rules with prefixes $00*, 01*, 1*, 10*$. We iteratively merge $00*$ and $01*$ to form a prefix $0*$, which is merged then with $1*$ into a common forward prefix. In this example, only one forward rule is needed for this set. As shown in Fig. 3, the forward rule covers the subtree with rules at every branch above the blue line.

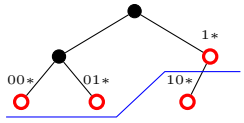



Fig. 3: Single-forward rule tree.

The set of resulting prefixes obtained by iterating the merging operation in $R$ and that are maximal for $>$ is called $MaxFwdMerges(R)$.

On a filtering path of length $n$, let $R_i$ be a rule subset placed in switch $s_i, 0 < i < n$. The potentially required forward rules in $s_{i+1}, i+1 \leq n$ after adding rules in $s_i$ are composed of successive $MaxFwdMerges$ results and they are called $PotentialFwds$. Considering $R_{i+1}$, only forward rules needed to resolve action conflicts with $R_{1,\ldots,i}$ are added in $s_{i+1}$. Thus, the forward rule set needed in switch $s_{i+1}$ is a subset of $PotentialFwds$. If a prefix covers only deny rules in $R$, packets matching this prefix in $s_i$ will not travel to the next switch. Such a prefix is not added to $PotentialFwds$ of $s_{i+1}$ since there will be no packet to match.

### B. Decomposition algorithm

Let us consider the binary tree shown in Fig. 4 which represents an ordered rule set. As an illustration example, each node contains one rule and we need to decompose the rule set along the path shown in Fig. 1 where each switch has a

maximal capacity of 6 rules. If we take into account the default rule in each switch, we need to find a set of 5 rules in order to completely fill the first switch. As depicted in Fig. 4, we select the set of nodes containing a total of 5 rules maximum (all nodes in red). After this selection step, all rules added to the switch will be removed from the binary tree. If some space remains in the switch, we will try to find other candidate rules.

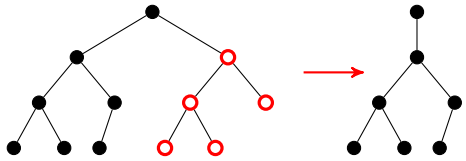


Fig. 4: Choosing a set of candidate rules from the binary tree to be placed in a switch.

The rule distribution is performed in one pass following a bin-focused approach and a **Minimum Bin Slack** heuristic [9]. Thus, each switch is filled as much as possible with a set of rules while trying to minimize the overhead caused by action conflicts with rules placed in previous switches. The required extra forward rules in switch $i$ for a rule set $R_i$ is defined as $NeededFwds(R_i, PotentialFwds) = \{fwd \in PotentialFwds \mid \exists r \in R_i$ such that $r >: fwd \wedge Action(r) =$ "$Deny$"$\}$.

## IV. Decomposition over a graph

In this section, we present a technique to enforce the same filtering policy across all paths in a network represented as a graph. Our algorithm for distributing a set of rules on a network uses a two-terminal series-parallel graph [10]. Then we will show how to handle more general two-terminal directed acyclic graphs.

Series-parallel graph are widely used in telecommunication networks [11], since the failure of a network part can be mitigated by using a parallel path, especially in critical applications with low tolerance to network paths failure. These types of networks can be updated in order to easily add or remove some parts by dividing the network into parallel or series compositions.

A series-parallel graph can be represented by a binary tree [12], where each internal node of the tree is a series or a parallel composition operation and each leaf is an edge of the graph. Fig. 6 shows a binary tree representation of a series-parallel graph. We define an S-component as an oriented path, that is either an edge or a series composition of edges (see Fig. 5). We can derive a more compact representation of series-parallel graphs by replacing maximal subtrees built solely from series operators by the S-components they represent. Hence, leaves are S-components in this alternative representation.

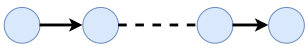


Fig. 5: S-component.

In the example of Fig. 6, edges $1 \rightarrow 2$ and $2 \rightarrow 4$ can be merged into one S-component, as edges $1 \rightarrow 3$ and $3 \rightarrow 4$.

In order to determine if our algorithm can find a solution given

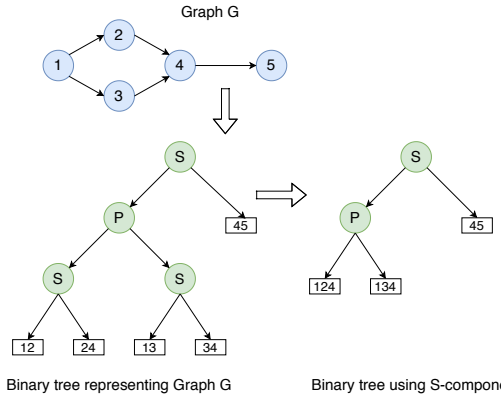



Fig. 6: Tree representation of a series-parallel graph.

a rule set $R$ to decompose, we need to try the decomposition on all paths of the directed network graph. We will then find the smallest rule set that can fit in all paths, taking into account the overhead in terms of forward rules and the compatibility between different paths that share some switches.

Our algorithm outputs a rule set to be deployed in a specific path from $s$ to $t$ using a representation of the graph as a binary tree with S-components. Each node of the binary tree has a set of rules, computed from the initial set, based on her parent being a series or parallel node.

Note that, in our approach and unlike the Optimized Big Switch (OBS) [4] technique, we do not create several rule table partitions for each path traversing an intersection switch. A packet will be processed by the same rule table at an intersection, regardless of the path it came from. This also facilitates policy updating, as the places where some specific rule occur are easier to localize.

Our technique can be applied to arbitrary two-terminal directed acyclic graphs or st-dags. Duffin [10] proved that a two-terminal st-dag is series-parallel if and only if it does not contain any subgraph homeomorphic to the so-called Braess graph. Our idea is to exploit the fact that merging some vertices in a subgraph homeomorphic to a Braess graph leads to a series-parallel one, after a few iterations.

## V. Evaluation

In our evaluation experiments of the different proposed algorithms, we rely on 12 rule sets available at [13]. These rule sets and ours are generated using ClassBench tool.
All our algorithms were implemented in Java with single-threaded programs. The machine used for this evaluation is a desktop computer with Intel core i7-7700 3.6-GHz CPU, 32 GB of RAM and running on the last version of Windows 10 operating system.

We define the overhead $OH$ in terms of additional forward rules placed on the switches of a path for a given rule set as follows:

$$OH = \frac{N_t - N_i}{N_i}$$

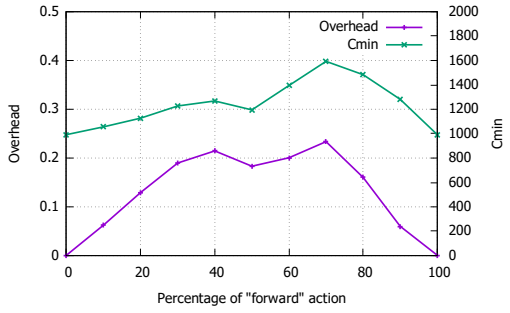
where

$N_t$ is the total number of rules used in a path (initial rule set plus extra forward rules).

$N_i$ is the number of rules in the initial rule set.

The overhead depends on the length of the path, capacity of the switches on the path and the diversity of rules' actions in the rule set.

The 12 rule sets used in our evaluations are the same as in [6]. In this evaluation all switches have the same capacity (tables size). The minimal capacity to find a solution by our distribution algorithm is defined as $C_{min}$. We decompose the rule sets using source IP address as a filtering field. Even with a synthetic rule set of 64000 random rules, no rule set takes more than 150ms to be decomposed and distributed. This processing time includes the tree building, rule set decomposition and rules distribution. In order to show the effect of the action field on the overhead $OH$ and the minimal capacity $C_{min}$, we use the rule set acl1, from the 12 rule sets [6], with a percentage of rules having "Forward" action between 0 to 100% by a step of 20%. The average values of $OH$ and $C_{min}$ are computed by running 10 simulations for each percentage value since the "Forward" actions will be distributed randomly among prefixes.




(a) acl1

Fig. 7: Effect of action field on overhead $OH$ and $C_{min}$ using acl1 rule set.

Fig. 7 shows the effect of the action field on both $OH$ and $C_{min}$ metrics. When a set of rules has very low action diversity, the overhead is very low since the number of conflicting actions is very small. The situation where the half of the rules has a "Forward" action introduces the most high overhead. We obtain similar results with the other rule sets.

Fig. 8 shows the overhead statistics using the 12 rule sets from Classbench while varying the path length. The overhead in the worst case is around 30% with 8 switches and 50% of accepting rules.

## VI. Conclusion and Future Work

in this paper, we proposed a new decomposition technique based on LPM for a single filtering field. We then introduced a novel rule set distribution algorithm for series-parallel network graphs. We have conducted a series of experiments on rule sets generated from Classbench. Our results show reasonable rule

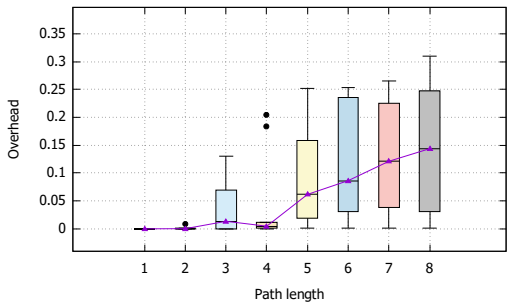



Fig. 8: Rule space overhead while distributing a rule set with a 50% of rules having Forward actions.

space overhead in worst-case scenarios for LPM. Our future work consists of tackling the rule update strategy problem. We also plan to implement and evaluate our solutions in real SDN environments.


## Acknowledgement

This work is supported by a CIFRE convention between the ANRT (National Association of Research and Technology) and the company NUMERYX Technologies.


## References


[1] A. Abboud, A. Lahmadi, M. Rusinowitch, M. Couceiro, A. Bouhoula, and M. Avadi, "Double mask: An efficient rule encoding for software defined networking," in *2020 23rd Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN)*, 2020, pp. 186–193.

[2] Y. Sun and M. S. Kim, "Tree-based minimization of TCAM entries for packet classification," in *2010 7th IEEE Consumer Communications and Networking Conference*, Jan 2010, pp. 1–5.

[3] A. Bremler-Barr and D. Hendler, "Space-Efficient TCAM-Based Classification Using Gray Coding," *IEEE Transactions on Computers*, vol. 61, no. 1, pp. 18–30, Jan 2012.

[4] N. Kang, Z. Liu, J. Rexford, and D. Walker, "Optimizing the "one big switch" abstraction in software-defined networks," 12 2013, pp. 13–24.

[5] Y. Kanizo, D. Hay, and I. Keslassy, "Palette: Distributing tables in software-defined networks," in *2013 Proceedings IEEE INFOCOM*, 2013, pp. 545–549.

[6] P. Chuprikov, K. Kogan, and S. Nikolenko, "How to implement complex policies on existing network infrastructure," in *Proceedings of the Symposium on SDN Research*, ser. ACM SOSR 18, New York, NY, USA, 2018.

[7] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: Enabling innovation in campus networks," *Computer Communication Review*, vol. 38, pp. 69–74, 04 2008.

[8] B. LeCun, T. Mautor, F. Quessette, and M.-A. Weisser, "Bin packing with fragmentable items: Presentation and approximations," *Theoretical Computer Science*, 01 2013.

[9] J. N. D. Gupta and J. C. Ho, "A new heuristic algorithm for the one-dimensional bin-packing problem," *Production Planning & Control*, vol. 10, no. 6, pp. 598–603, 1999.

[10] R. Duffin, "Topology of series-parallel networks," *Journal of Mathematical Analysis and Applications*. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0022247X65901253

[11] P. Flocchini and F. L. Luccio, "Routing in series parallel networks," *Theory of Computing Systems*, 2003. [Online]. Available: https://doi.org/10.1007/s00224-002-1033-y

[12] J. Valdes, R. E. Tarjan, and E. L. Lawler, "The recognition of series parallel digraphs," in *Proceedings of the Eleventh Annual ACM Symposium on Theory of Computing*, ser. STOC 79. New York, NY, USA: Association for Computing Machinery, 1979, p. 112.

[13] *Code for Palette, OBS and OneBit simulations*, 2017. [Online]. Available: https://github.com/distributedpolicies/submission