Optimizing Rules Placement in OpenFlow Networks: Trading Routing for Better Efficiency

Xuan-Nam Nguyen, Damien Saucez, Chadi Barakat, Thierry Turletti
INRIA Sophia Antipolis Méditerranée, France
{FirstName.LastName}@inria.fr

ABSTRACT

The idea behind Software Defined Networking (SDN) is to conceive the network as one programmable entity rather than a set of devices to manually configure, and OpenFlow meets this objective. In OpenFlow, a centralized programmable controller installs rules onto switches to implement policies. However, this flexibility comes at the expense of extra overhead as the number of rules might exceed the memory capacity of switches, which raises the question of how to place most profitable rules on board. Solutions proposed so far strictly impose paths to be followed inside the network. We advocate instead that we can trade routing requirements within the network to concentrate on where to forward traffic, not how to do it. As an illustration of the concept, we propose an optimization problem that gets the maximum amount of traffic delivered according to policies and the actual dimensioning of the network. The traffic that cannot be accommodated is forwarded to the controller that has the capacity to process it further. We also demonstrate that our approach permits a better utilization of scarce resources in the network.

Categories and Subject Descriptors

C.2.3 [**Network Operations**]: Network Management

Keywords

Software-Defined Networking; OpenFlow; Rule Placement

1. INTRODUCTION

Software Defined Networking (SDN) [13] paradigm is aiming to facilitate the task of network management by transforming switches into programmable entities. Programmability is performed in a centralized way, which increases network flexibility and eases the introduction of new functions and services across the network. OpenFlow [9] is a communication protocol enabling SDN, which has been implemented in commodity switches and starts to be deployed in production networks [6]. SDN in general and OpenFlow in particular enable fine-grained flow-level traffic control for various management tasks (e.g., rate limitation, access control, accounting). A logically centralized controller is responsible of the allocation of the decision tables in switches by installing on them the appropriate forwarding rules.

HotSDN'14, August 22, 2014, Chicago, IL, USA.
Copyright is held by the owner/author(s). Publication rights licensed to ACM.
ACM 978-1-4503-2989-7/14/08 ...$15.00.
http://dx.doi.org/10.1145/2620728.2620753.

However, many controller platforms [1, 4] still require network programmers to manually transform high-level policies into low-level rules and to decide where to install them. This is unfortunately a complex task giving the large number of flows to handle and the diversity of constraints and strategies to be accounted for. The limited available memory on switches and the capacity of network links are typical constraints. Indeed, the memory limitation on switches restricts the number of rules that can be installed and makes more challenging the process of mapping high-level policies into low-level rules. In fact, OpenFlow rules are more complex than forwarding rules in normal routers, as they support flexible matchings and sophisticated actions rather than simply forwarding based on IP prefixes. According to today's technology, a commodity OpenFlow switch can support up to tens of thousands of rules [14], far below what is required for an exhaustive network management [10]. The same problem exists in software switches for which performances are limited by CPU caches. This urges the need for optimized and automated solutions for rule placement able to select and place the most profitable rules across the network.

Recent studies have explored the problem of OpenFlow rules selection and placement [10, 2, 7, 8, 16], as detailed in Section 2. Some of them as [2] propose local solutions to switches for a better flow management. Other studies [7, 8] advance a network-wide problem but still require the operator to define for the different flows the treatment they should expect from the network in terms of routing. The role of the network-wide optimization is then to minimize the volume of resources required to handle the flows efficiently. This approach imposes on the optimizer and the centralized controller limitations on the way flows should be treated inside the network (i.e., the exact path to follow). Furthermore, existing solutions focus on the minimum resources required, thus making the implicit assumption that all flows can always finally achieve their desired processing. In this paper we advocate a more general approach. Our new approach differs in two main aspects. On one side, it only limits the constraints imposed on the optimizer to the simple one of respecting the endpoint policy as long as the traffic entering the network at some specific point leaves it at the desired egress points. This assumes that the SDN network is reaching its objective whatever is the path followed across the network (e.g., the cost of using internal links is low, the network is of low diameter, etc.). As a consequence, paths inside the network can be adapted for the best utilization of network resources. On the other hand, we optimize the rule placement for given network resources and traffic demand. Differently speaking, we do not seek the best dimensioning of the network that accommodates actual traffic, but rather get the maximum from the actual dimensioning of the network given the traffic demand. This implies that part of the traffic might not get the desired processing by the network: we suppose for this part that the network switches are equipped with a default forwarding plane carrying it into one or more controllers inside the network where it can be later processed (e.g., tunneled to its final destination or dropped). In a complement to the first aspect, the target here is to minimize the volume of the traffic following the default behavior.

Through these two aspects, we present a novel way to consider SDN networks composed of OpenFlow switches, yielding further flexibility and better efficiency. In the following we present the main lines of the approach and comment on a possible modeling work that can be carried out to implement the network-wide optimization. We then discuss the efficiency gain we obtain by trading routing and dimensioning in favor to the endpoint policy. We finally open the discussion to general questions brought by our vision.

2. STATE OF THE ART

One of the major challenges of SDN is to raise the level of abstraction with the objectives of easing network management and facilitating the development, debugging and deployment of new applications. Frenetic [3] and Procera [15] are two programming languages for OpenFlow networks that provide a rich environment and operators to transform high-level policies into switch-level rules.

Recently, several studies [2, 11, 5] have advocated to aggressively use wildcard rules to limit interactions between switches and controller and to minimize the rule space consumption on switches. Among these studies, DevoFlow [2] handles short flows using wildcard rules, whereas DomainFlow [11] splits the network into two domains: one using wildcard rules and the other one exact matching rules. In SwitchReduce [5], all rules that have identical actions are compressed into a wildcard rule, except at the first hop switch.

On the rule placement side, DIFANE [16] classifies rules and caches the most important ones at some special devices, named authority switches. Then, ingress switches redirect unmatching packets towards these authority switches, which allows reducing the load on the controller as well as the number of rules required to be stored on ingress switches. vCRIB [10] advocates to put rules in both hypervisors and switches to achieve a good trade-off between performance and resource usage. All these techniques require additional devices to be used.

Palette [8] and OneBigSwitch [7] take into account both the endpoint policy and the routing policy to produce the aggregated rule sets and to decide where to install them in the network. More precisely, Palette decomposes the large endpoint policy table into sub-tables and inserts them on the switches so that each packet traverses all the tables at least one. In this manner, it is equivalent to a single lookup in the original large endpoint policy table. However, as shown in [7], Palette leads to suboptimal solutions because it enforces shortest path routing and mandates that all network paths fully satisfy the endpoint policy. Palette thus is not able to benefit from all available switches in the network when shortest paths span few switches. OneBigSwitch outperforms Palette by exploiting more paths and enforcing part of the endpoint policy along each path.

These solutions mainly target the network provisioning problem, and they can efficiently place rules to satisfy the endpoint policy while minimizing resources. However, they are not appropriate for scenarios where there are not enough resources to satisfy the whole endpoint policy. We present examples of such use cases in Section 3.3. The aspects covered by our approach, including mainly the placement of the most profitable rules under constrained resources, and the relaxation of the routing policy, are novel and allow better efficiency and more flexible network programming.

3. OUR VISION

The approach followed so far in the literature ([8, 7]) to solve the OpenFlow rules selection and 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 such that these paths are respected. Sometimes, imposing paths is necessary to ensure performance and cost levels, however, in most situations where OpenFlow would be used, the impact of the exact path followed by traffic within the network is negligible (e.g., cost or delay) as long as the traffic can reach its destination. Therefore, we advocate that while it is essential to respect the *endpoint policy* (i.e., reach the desired set of egress points for set of packets matching a pattern) to ensure that the traffic is delivered to its destination, we can relax the *routing policy* (i.e., not imposing on the traffic to follow a specific path through the network as long as there are no loops). Relaxing the routing policy improves the path diversity, which increases the network capacity as resources not used when the path is imposed become usable without the constraint; the hidden cost being a higher complexity of selecting and placing rules as the space of potential solutions to explore is larger.

To take into consideration the default path forwarding, to satisfy the endpoint policies and also to avoid forwarding loops, we propose to forward all packets from the same ingress point over the same default arbitrary path, until they are possibly diverted to a path that leads them to one of their suitable egress points. In practice, the default path can be the one that leads to the controller. However, as the routing policy is relaxed, the actual path to follow to reach a suitable egress point is not imposed. Therefore, nothing prevents a packet to follow the default path until it is shunted with a path going toward the right egress point.

In view of this, solving the allocation and placement problem is equivalent to finding the switches where to shunt the default paths with paths to egress points. The role of the optimization herein is to maximize the total value of traffic that can be sent to its suitable egress point with regard to the available resources, which is an important factor to account for when resources are scarce. This is a key difference of our approach compared to solutions in the literature that

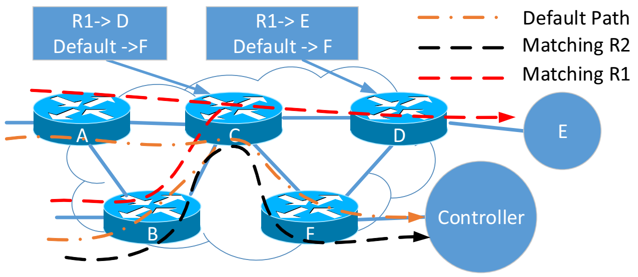

Figure 1: Example of the routing policy relaxation

try to find the minimum volume of resources necessary to accommodate all traffic.

Figure 1 illustrates the rule placement problem in a simple scenario. Each switch only has one available flow entry, beside the default entry. We have two different flows `R1` and `R2` whose suitable egress point is `E` for both and traffic for `R1` being larger than traffic for `R2` and in case of scarce resource, the policy is to maximize the amount of traffic readily delivered to suitable egress points. With our solution, we first let all packets follow the default path on switches `A` and `B`. Then, packets of `R1` are deflected to the preferred egress point using specifically installed rules on switch `C` ($R1 \rightarrow D$) and switch `D` ($R1 \rightarrow E$). However, as there is no more room on switch `C`, neither on switch `D`, rules for `R2` cannot be installed and packets of `R2` are eventually delivered to the controller, via the default path, for further processing.

3.1 Maximizing traffic satisfaction

In the following we propose an example of integer linear programming model that can be used to compute the best allocation of rules when the routing policy is relaxed. To highlight the interest of using default paths, we optimize the rule placement for a given network resource and traffic demand, which would typically be the case when migrating a network to OpenFlow. Our optimization problem must then take into account switch memory and link capacity constraints which implies that part of the traffic might not be delivered directly to its desired egress points but would go to the controller instead. Maximizing the traffic satisfaction for a given resource is the dual problem of finding the best network provisioning to accommodate traffic as envisioned by Palette [8] or OneBigSwitch [7], and nothing prevents relaxing the routing policy for that problem as well.

Before Section 3.2, we define the following terms that are used throughout the text. A *matching pattern* is the combination of meta information related to packets (e.g., destination IP address, protocol, origin) that switches use to determine the action (e.g., forward, drop, encapsulate) to be performed on packets. In the following, the symbol R denotes the set of matching patterns and $\forall r \in R, E(r) \subseteq E$ is the set of egress points where it is suitable to forward packets matching the pattern r. A *flow* is the set of all packets that match the same matching pattern and an *iFlow* is the subset of packets that belong to the same flow and that share the same ingress point. The traffic workload of the network is modeled as the set of iFlows F in conjuction with p_f that denotes the packet rate of the iFlow $f \in F$. By abuse of notation, the matching pattern associated to an iFlow $f \in F$ is denoted r_f and $w_{f,e}$ stands for the weight the endpoint policy gives to making packets of iFlow f leaving the network via the egress point $e \in E(r_f)$. A network is composed of a logically centralized controller and set of interconnected switches S. A *path* is an ordered sequence of switches separating two points in the network and $Successor(f, e, s)$ defines the path to follow in case the rule for $f \in F$ is installed on $s \in S$ and that egress point $e \in E(r_f)$ has been selected as egress point for the iFlow. Similarly, $DefPath(f)$ is the default path that packets from an iFlow $f \in F$ would follow if they could not be matched, starting to the ingress point of the iFlow. Finally, the way paths are computed is let to the discretion of the network operator as long as each path is loop free.

3.2 Problem Formulation

Our objective is to maximize the value of the traffic that respects the endpoint policy (i.e., delivered to its most appropriate egress points) and we can write our objective function as follows:

$$max \sum_{f \in F} \sum_{e \in E(r_f)} w_{f,e}\, y_{f,e} \tag{1}$$

where the Boolean variable $y_{f,e}$ indicates whether packets of iFlow f are delivered to egress point e.

It is thus necessary to build a Boolean placement $|R|$-by-$|E|$-by-$|S|$ matrix $A = (a_{r,e,s})$, where $a_{r,e,s}$ indicates whether a rule must be installed on switch $s \in S$, so that packets associated to the matching pattern $r \in R$ are delivered to egress point e ($a_{r,e,s} = 1$) or not ($a_{r,e,s} = 0$).

To take into consideration the default path forwarding, the endpoint policies and also to avoid forwarding loops, acceptable solutions to this rule placement problem must satisfy the following constraints:

$$\forall f \in F, \forall e \in E(r_f) : y_{f,e} \in \{0, 1\} \tag{2}$$
$$\forall r \in R, \forall e \in E(r), \forall s \in S : a_{r,e,s} \in \{0, 1\} \tag{3}$$
$$\forall f \in F : 1 \geq \sum_{e \in E(r_f)} y_{f,e} \tag{4}$$
$$\forall f \in F, \forall e \notin E(r_f) : y_{f,e} = 0 \tag{5}$$
$$\forall f \in F, \forall e \in E(r_f) : y_{f,e} \leq \sum_{s \in DefPath(f)} a_{r_f,e,s} \tag{6}$$
$$\forall f \in F, \forall e \in E(r_f), \forall s \in DefPath(f) : y_{f,e} \geq a_{r_f,e,s} \tag{7}$$
$$\forall r \in R, \forall s \in S : \sum_{e \in E(r)} a_{r,e,s} \leq 1 \tag{8}$$
$$\forall r \in R, \forall s \in S, \forall e \notin E(r) : a_{r,e,s} = 0 \tag{9}$$
$$\begin{aligned}&\forall f \in F, \forall e \in E(r_f),\\ &\quad \forall s \in DefPath(f), \forall n \in Successor(f, e, s) :\\ &\qquad (a_{r_f,e,s} - a_{r_f,e,n}) \times \Delta(f, e, n) \leq 0\end{aligned} \tag{10}$$

Constraints (2) and (3) ensure that $y_{f,e}$ and $a_{f,e,s}$ are binary variables. Constraints (4) and (5) guarantee that packets of the iFlow f are delivered to at most one egress point and that this egress point satisfies the endpoint policy $E(r_f)$. For the sake of generality, two iFlows with the same matching pattern r can be delivered to two different egress points. All iFlows that cannot be delivered to an egress point are directed to the controller and constraint (6) guarantees that if iFlow f is deflected to the egress point e, then a rule allowing this is installed on at least one switch on the default path of iFlow f ($y_{f,e} = 1 \Rightarrow \exists s \in DefPath(f)$ such that $a_{r_f,e,s} = 1$). Constraint (7) indicates that if an iFlow

f is not delivered to an egress point e, no rule for the iFlow is installed on any switch on the default path ($y_{f,e} = 0 \Rightarrow a_{r_f,e,s} = 0, \forall s \in \mathit{DefPath}(f)$).

Constraints (8) and (9) ensure that a matching pattern r has at most one entry on the switch s, and that packets matching r are forwarded toward one of their egress points $E(r)$.

Constraint (10) avoids forwarding loops by imposing that once a rule is installed on a switch, it must also be installed on all successors of that switch toward the selected iFlow's egress point e (except switches where it is able to use the default rule for forwarding packets for f towards egress point e, which is denoted by $\Delta(f, e, n) = 0$) ($a_{r_f,e,s} = 1 \Rightarrow a_{r_f,e,n} = 1$ if $\Delta(f, e, n) = 1, \forall n \in \mathit{Successor}(f, e, s)$).

Switch memory limitation: To account for switch memory limitation, we denote C_s as the number of rules that can be installed in the memory of any switch $s \in S$. The following constraint takes memory limitations into account:

$$\forall s \in S : \sum_{r \in R} \sum_{e \in E(r)} a_{r,e,s} \leq C_s \tag{11}$$

One can extend the model further to allow rules to be aggregated on switches and hence reduce their memory usage. However, for room issues, we let the reader consult the details of the method on our companion technical report for this paper on [12].

Link capacity limitation: The concept of leveraging a default path implicitly assumes that the internal network is provisioned well enough to support the traffic load. However, it does not mean that egress links are all over-provisioned and care must be taken to (1) favor some egress points and (2) not overload egress links. The first objective is assured by the weighted factor in Eq.(1). To account for the capacity of egress links and not overload them, let B_e be the capacity of the link for egress $e \in E$. The constraint to not overload egress links is then:

$$\forall e \in E : \sum_{f \in F} y_{f,e} \cdot p_f \leq B_e. \tag{12}$$

If the capacity of the link connected to the controller is limited, the following constraint that only accounts for traffic delivered to the controller can also be added:

$$\sum_{f \in F} \left(1 - \sum_{e \in E(r_f)} y_{f,e} \right) \cdot p_f \leq B_*. \tag{13}$$

Solving the above problem returns the rule placement solution, i.e., which rule and where to install in the network in order to maximize traffic satisfaction. We are currently working on extending our model to account for bandwidth limitation for links inside the network. However, the adaptation is not straightforward as it requires to track the path followed by the different iFlows that might be partially shared with the default path.

3.3 Use cases

The optimization model that we formalized in Section 3.2 is designed to cover a broad range of networking use cases

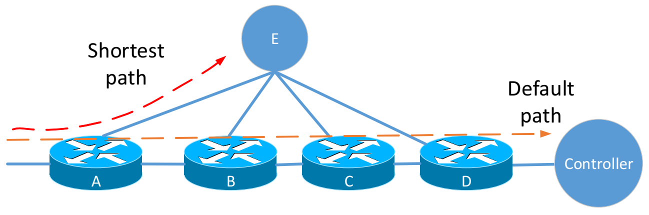

Figure 2: Example topology

thanks to the level of abstraction of the objective function. In this section we present three common use cases that can be solved with our model.

Load balancing: For reliability and efficiency reasons, networks are commonly connected to several service providers, or services are running on several servers such that traffic load can be balanced between several points. When rules aggregation is enabled, our optimization problem can be used to achieve load balancing. To do so, and without loss of generality, an iFlow to be load balanced among several egress points must be artificially decomposed into multiple iFlows (i.e., with different matching patterns) that are assigned the same set of egress points but with different weights, such that each iFlow gives better value for a particular egress point. The exact way iFlows can be decomposed is out of the scope of this paper. For example, if the traffic to `192.0.2.0/24` must be equally balanced between egress e_1 and e_2, it can be decomposed into iFlows f_1 and f_2 such that r_{f_a} is `192.0.2.0/25` and r_{f_2} is `192.0.2.128/25` and weights can be set as follows: $\{w_{f_1,e_1} = 2, w_{f_1,e_2} = 1\}$ and $\{w_{f_2,e_1} = 1, w_{f_1,e_2} = 2\}$.
Load balancing within the network is also possible as if two iFlows share the same egress point, the path followed by packets can still be different. With our solution, the load is not only balanced among egress points, but also among different paths in the network. It is also worth to notice that weights can be adapted so that some egress points get more traffic load than others.

Access Control List: It is common in enterprise networks to use access control lists to filter out unwanted traffic. We can use our method to implement them by distributing the firewall rules among the switches in the network. In this case, the egress points associated to iFlows of packets to be dropped are all the switches and the *Successor(f, e, s)* function must be defined to always return s (i.e., the switch where the rule is installed first). Moreover, the action associated to an installed rule for these iFlows is to drop matching packets.

Machine to machine communications: Our method can be used to install the forwarding plane in datacenters when the traffic pattern between virtual machines can be estimated. This problem is casted in our ILP by assigning the destination server as the egress point for the traffic sent between two machines. To spread the traffic load inside the network, the load balancing technique described here above can be applied as well.

4. EARLY STUDY

This section provides a brief study of our approach to show the potential of relaxing the routing policy. We show with an illustrative synthetic topology that the optimization prob-

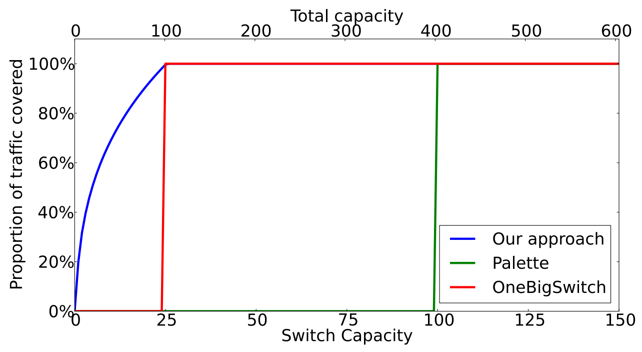

Figure 3: Proportion of traffic covered with different rules allocation schemes

lem we propose is able to explore a larger space of solutions than Palette [8] or OneBigSwitch [7].

The objective of Palette and OneBigSwitch is to seek for the best provisioning of the network that accommodates with the traffic and policies. However, as our idea is to relax routing constraints to make the network a true black box, it becomes natural to envision the dual problem that consists in finding the best allocation for a given network infrastructure, which would typically be the case upon migration to OpenFlow. Differently speaking, the first approach determines the minimum set of resources necessary to accommodate with traffic while our approach offers to accommodate as much traffic as possible for the budget of available resources.

To compare the dual approaches, we use the topology illustrated in Figure 2 and determine the proportion of traffic that can be satisfied for different levels of available resources in the network.

We consider four core routers, all identical, that form a line topology and that are all directly connected to the same egress point E where some non-aggregatable traffic must be sent eventually. All that traffic is originated from the same end of the chain and the controller is located at the other end of the chain. The chain thus forms the default path toward the controller.

Figure 3 illustrates the proportion of satisfied traffic (i.e., proportion of traffic delivered to E, assuming that traffic rate follows a Zipf distribution of decay parameter 0.7), for different switch memory capacities in the case of 100 non-aggregatable traffic flows. In our example case, one can prove that a total of at least 100 rule entries are required globally in the network to deliver 100% of the traffic without intervention of the controller. Palette enforces the shortest path policy, i.e., all the packets must follow the path $A \rightarrow E$, meaning that memory for at least 100 rule entries must be available on switches to solve the allocation problem, raising the total memory capacity of the network to 400 rule entries which is sub-optimal. On the contrary, by decoupling the network into paths and placing rules along each of the four possible paths ($A \rightarrow E, A \rightarrow B \rightarrow E, A \rightarrow B \rightarrow C \rightarrow E, A \rightarrow B \rightarrow C \rightarrow D \rightarrow E$), OneBigSwitch manages to optimally satisfy traffic allocation with no more than 25 rules per switch (100 rule entries in total in the network). Our scheme is also able to optimally satisfy the whole traffic as long as there are at least 100 rule entries available in the network. However, as opposed to the other schemes, our optimization problem allows to deliver some proportion of traffic even if the available memory is not sufficient on switches. Our evaluation shows that our scheme is complementary to OneBigSwitch as it permits to explore the case of under-provisioned networks.

5. DISCUSSION

Solutions proposed so far to address the OpenFlow rule selection and placement problem assume strict endpoint policy that exactly states where packets must leave the network. Moreover, they also assume the known routing policy that disclose the paths that packets should follow to reach their egress points. In this paper, we advocate that in most of OpenFlow networks, the routing policy can be ignored as only the correct delivery of packets matters, regardless of the path that they follow. We also prescribe to leverage the usage of the controller to maximize traffic satisfaction when network resources are scarce. We propose an Integer Linear Programming variant to support these claims. More precisely, we aim to find a rule allocation matrix that maximizes the total value of the traffic that satisfies the endpoint policy. The unsatisfied traffic is delivered to the controller for further processing (e.g., tunneled to right egress points or simply dropped). A brief comparison with the current state of the art shows that our solution achieves at least as good traffic satisfaction as the other solutions when resources are sufficient and outperforms them in case of scarcity of resources (e.g., memory, link capacity).

Our optimization problem allows partial satisfaction of the endpoint policy in constrained network environments and also traffic load balancing between several egress points. However, this flexibility comes at a potential intensive computation cost (one can prove the maximizing traffic satisfaction problem to be NP-hard by reducing it to the Knapsack problem). Nevertheless, in most situations, operators are willing to reduce the overhead of interacting with the controller. If such is the case, valuate the traffic by its rate or volume is a good approach. Moreover, if the traffic follows a long-tail distribution, a linear complexity *greedy* heuristic to solve the optimization problem is a promising candidate as it can favor large flows, which rapidly bring value to satisfy the endpoint policy, at the expense of small flows that bring little value overall. As future work, we are exploring ways to solve the rules placement problem with online algorithms in order to automatically react to network and traffic changes.

Our vision is that one can trade the routing policy to increase efficiency when resources are limited, as in most OpenFlow-capable networks, the exact path followed by packets is not important. However, the location of the controller in the topology might still have an impact on the overall performances and the interaction with our proposition merits to be studied in details.

Acknowledgments

This work is partly funded by the French ANR under the "ANR-13-INFR-013" project and by the European "KIC EIT ICT Labs Activity 13153" on Software-Defined Networking.

6. REFERENCES

[1] Floodlight.
http://www.projectfloodlight.org/floodlight/.

[2] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee. DevoFlow: scaling flow management for high-performance networks. *SIGCOMM CCR*, 41(4):254–265, Aug. 2011.

[3] N. Foster, R. Harrison, M. J. Freedman, C. Monsanto, J. Rexford, A. Story, and D. Walker. Frenetic: A Network Programming Language. *ACM SIGPLAN*, 46(9):279–291, Sept. 2011.

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

[5] A. Iyer, V. Mann, and N. Samineni. Switchreduce: Reducing switch state and controller involvement in openflow networks. In *IFIP Networking Conference, 2013*, pages 1–9, May 2013.

[6] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, et al. B4: Experience with a globally-deployed software defined WAN. In *SIGCOMM*, pages 3–14. ACM, 2013.

[7] N. Kang, Z. Liu, J. Rexford, and D. Walker. Optimizing the 'One Big Switch' Abstraction in Software-Defined Networks. In *CoNEXT*. ACM, Dec. 2013.

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

[9] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. OpenFlow: enabling innovation in campus networks. *SIGCOMM CCR*, 38(2):69–74, Mar. 2008.

[10] M. Moshref, M. Yu, A. Sharma, and R. Govindan. vcrib: Virtualized rule management in the cloud. In *USENIX HotCloud*, pages 23–23, Berkeley, CA, USA, 2012.

[11] Y. Nakagawa, K. Hyoudou, C. Lee, S. Kobayashi, O. Shiraki, and T. Shimizu. Domainflow: Practical flow management method using multiple flow tables in commodity switches. In *CoNEXT*, pages 399–404. ACM, 2013.

[12] X.-N. Nguyen, D. Saucez, C. Barakat, and T. Turletti. Optimizing rules placement in OpenFlow networks: trading routing for better efficiency. Technical report, INRIA, http://www-sop.inria.fr/members/Xuan-Nam.Nguyen/Publications/namoptimizing.pdf, 2014.

[13] B. Nunes, M. Mendonca, X. Nguyen, K. Obraczka, and T. Turletti. A survey of software-defined networking: Past, present, and future of programmable networks. *IEEE Comm. Surveys & Tutorials*, PP(99):1–18, 2014.

[14] B. Stephens, A. Cox, W. Felter, C. Dixon, and J. Carter. PAST: scalable ethernet for data centers. In *CoNEXT*, pages 49–60. ACM, 2012.

[15] A. Voellmy, H. Kim, and N. Feamster. Procera: a language for high-level reactive network control. In *HotSDN*, pages 43–48. ACM, 2012.

[16] M. Yu, J. Rexford, M. J. Freedman, and J. Wang. Scalable flow-based networking with DIFANE. *SIGCOMM CCR*, 41(4), Aug. 2010.