Too many SDN rules? Compress them with MINNIE

M. Rifai[†], N. Huin[*†], C. Caillouet [*†], F. Giroire [*†], D. Lopez-Pacheco[†], J. Moulierac[*†], G. Urvoy-Keller[†]
[*] Inria Sophia Antipolis
[†] University of Nice Sophia Antipolis, CNRS
I3S, UMR 7271, 06900 Sophia Antipolis, France
Email: mrifai@i3s.unice.fr

***Abstract*—Software Defined Networking (SDN) is gaining momentum with the support of major manufacturers. While it brings flexibility in the management of flows within the data center fabric, this flexibility comes at the cost of smaller routing table capacities. In this paper, we investigate compression techniques to reduce the forwarding information base (FIB) of SDN switches. We validate our algorithm, called MINNIE, on a real testbed able to emulate a 20 switches fat tree architecture. We demonstrate that even with a small number of clients, the limit in terms of number of rules is reached if no compression is performed, increasing the delay of all new incoming flows. MINNIE, on the other hand, reduces drastically the number of rules that need to be stored with a limited impact on the packet loss rate. We also evaluate the actual switching and reconfiguration times and the delay introduced by the communications with the controller.**

I. INTRODUCTION

In classical networks, routers compute routes using distributed routing protocols such as OSPF (Open Shortest Path First) [1] to decide on which interfaces packets should be forwarded. In the Software Defined Networks (SDN) paradigm, one or several controllers take care of route computations and routers become simple forwarding devices. When a packet arrives with a new destination for which no routing rule exits, the router[1] contacts a controller that provides a route to the destination. Then, the router stores this route as a rule in its SDN table and uses it for next incoming matching packets. This separation of the control plane from the forwarding plane allows a smoother control over routing and an easier management of the routers.

However, SDN capable forwarding devices use Ternary Content-Addressable Memory (TCAM) to store their routing table. This efficient memory is expensive and thus limits the maximum size of the table. Also, SDN rules can be more complex than classical routing rules. A typical switch supports in the order of a thousand 12-tuple flows; the actual number ranges from 750 to 4000 [2]. Undoubtedly, emerging switches will support larger rule tables [3], but TCAMs still introduce a fundamental trade-off between rule-table size and other concerns like cost and power. The maximum size of routing tables is thus limited and represents an important concern for the deployment of SDN technologies. This problem has been addressed in previous works, as discussed in Section II, using different strategies, such as routing table compression [4], or distribution of forwarding rules [5].

This work has been partially supported by ANR VISE and ANR program Investments for the Future under reference ANR-11-LABX-0031-01.

[1]In the following, we make no distinction between routers/switches, packets/frames and routing/forwarding tables using these terms in their general sense.

In this work, we examine a more general framework in which *table compression* using wildcard rules is possible. Compression of SDN rules was discussed in [4]. The authors propose algorithms to reduce the size of tables, but only by using a default rule. We consider here stronger compression rules in which any packet header fields may be compressed. Considering *multiple field aggregation* is an important improvement as it allows a more efficient compression of routing tables, better routing methodology using in particular load-balancing and the introduction of quality of service policies. In the following, we focus on compression of rules based on sources and destinations, but other fields may be considered such as ToS (Type of Service) field or transport protocol.

In this paper, we consider the problem of dynamically routing traffic demands inside a data center network using SDN technologies. In Section III, we provide the routing algorithm satisfying the link capacity constraints and the compression algorithm satisfying the routing table size constraints of the different forwarding devices. We validate our algorithms with a testbed composed of SDN hardware as explained in Section IV. We study different metrics, namely the delay introduced by the communications with the controller, the potential increase of fault rate due to the handling of the dynamic routing and the load of the controller with and without compression. Our results (Section V and VI) show that we are able to minimize the number of entries in the switches, while successfully handling client's dynamics and keeping the network stability. Finally, we close this document in Section VII by providing concluding remarks and future works.

II. RELATED WORK

To support a vast range of network applications, SDN has been designed to apply flow-based rules, which are more complex than destination-based rules in traditional IP routers. For instance, access-control requires matching on source, destination IP addresses, port numbers and protocol [6] whereas a load balancer may require matchings only on source and destination IP prefixes [7].

These complicated matchings can be well supported using TCAM since all rules can be read in parallel to identify the matching entries for each packet. However, as TCAM is expensive and extremely power-hungry, the on-chip TCAM size is typically limited. Many existing studies in the literature have tried to address this limited rule space problem. For instance, the authors in [8] and [9] try to compact the rules by reducing the number of bits describing a flow within the switch by inserting a small tag in the packet header. This solution is complementary to ours, however, it requires a change in:

(i) packet headers and (ii) in the way the SDN tables are populated. Also, adding an identifier to each incoming packet is hard to be done in the ASICs since this is not a standard operation, causing the packets to be processed by the CPU (a.k.a. the slow-path), strongly penalizing the performance of a forwarding device and the traffic rate. Another approach is to compress policies on a single switch. For example, the authors in [10], [11], [12] have proposed algorithms to reduce the number of rules required to realize policies on a single switch.

To the best of our knowledge, the closest papers to our work are from [13], [5], [14], [15], [16]. In [5] the authors model the problem by setting constraints on the nodes that limit the maximum size of routing table. The rules are spread between the switches to avoid QoS degradation in case of full forwarding tables. However, no compression mechanisms are proposed. For the case of [14], we note that the replacement of a routing policy, to follow the dynamicity of the network, can be hard since it implies to rebuild the forwarding table in (potentially) several switches. In [15], the authors propose OFFICER. In OFFICER, a default path is created for all the communications, and later, some deviations are introduced from this path using different policies to reach the destination. According to the authors, the Edge First (EF) strategy, where the deviation is placed to minimize the number of hops between the default path and the target, offers the best trade-off between the required QoS and forwarding table size. Note however, that applying this algorithm could unnecessarily penalize the QoS of flows when the switches' forwarding tables are rarely full. In [16] the authors suggest XPath which identifies end-to-end paths using path ID and then compress all the rules and preinstall the necesary rules into the TCAM memory. [4] addresses the problem of compressing routing tables using default rule only in case of Energy-Aware Routing. We extend this solution by considering other types of compression.

In this work we study a new and original way to compress the rules in SDN tables using aggregation by the source and destination. This kind of compression allows to keep the same paths as the original ones keeping the length of paths unchanged in order to guarantee the best quality of service. Last, but not least, our proposition can be easily deployed in real SDN networks, as shown in Section IV.

III. Modeling of the problem and Description of Minnie algorithm

We represent the network as a directed graph $G = (V, A)$. A vertex is a router and an arc represents a link between two routers. Each link has a maximum capacity and the number of rules of a router is limited by the size of its routing table. For a set of demands $\mathcal{D}$, a routing solution consists in assigning to each demand a path in a way that the capacity constraints and the table size constraints are respected.

We define a routing rule as a triplet (s, t, p) where s is the source of the flow, t its destination and p the output port in the network. To aggregate the different rules, we use *wildcard rules* that can merge rules by source (i.e. $(s, *, p)$), by destination (i.e. $(*, t, p)$) or both (i.e. $(*, *, p)$, the default rule). Figure 1 shows an example of a routing table and its compressed version using different strategies. Rules have priorities over one another (based on their ordering) in case multiple rules correspond to a flow. For example, in the solution with the minimum number of rules (Fig. 1(e)), rule $(1, *)$ has to have a higher priority than rule $(*, 4)$ in the table, otherwise the flow $(1, 4)$ would be routed through Port-4, which is not the routing decision taken in Fig. 1(a).

A. Minnie: *Routing phase*

We propose an efficient routing heuristic where the flows are spread over the network to avoid overloading a link or a table using a shortest-path algorithm with an adaptive metric.

For every demand between two nodes s and t with a charge d, we compute a route by finding a shortest path in a digraph G' representing our network. G' is a subgraph of G where an arc (u, v) is removed if its capacity is less than d or if the router u is full and does not contain any wildcard rule for (s, t, v) (where v represents also the output port of u towards v). The weight w_{uv} of a link depends on the flow on the link and the table's usage of router u. We note w^c_{uv} the weight corresponding to the link capacity and w^r_{uv} the weight corresponding to the rule capacity. They are defined as follows:

$$w^c_{uv} = \frac{\mathcal{F}_{uv} + d}{C_{uv}}$$

where C_{uv} is the capacity of the link (u, v) and $\mathcal{F}_{uv}$ the total flow on (u, v). The more the link is used, the heavier the weight is, which favors the use of lower loaded links allowing load-balancing. And

$$w^r_{uv} = \begin{cases} \frac{|R_u|}{S_u} & \text{if } \nexists \text{ wildcard rule for } (s, t, v) \\ 0 & \text{otherwise} \end{cases}$$

where S_u is the maximum table size of router u and R_u is the set of rules for router u. The weight is proportional to the usage of the table.

The weight w_{uv} of a link (u, v) is given by:

$$w_{uv} = 1 + 0.5 * w^c_{uv} + 0.5 * w^r_{uv}$$

Once a path is found for a demand, for each arc (u, v) of the path, we add the rule (s, t, v) to the router u if no corresponding wildcard rule exists. If the table is full, we **compress** it using the algorithm described previously. We also reduce the capacity of the arc by d. If no path is found, this means that the links are overloaded ; the demand is not routed leading to packet loss.

B. Minnie: *Compression phase*

Since the compression of a table with wildcard rules is NP-Hard [17], we propose an efficient heuristic to compress tables using aggregation by source, by destination and by the default rule. The heuristic computes three compressed routing tables and, then, chooses the smallest one.

For the first one (Fig. 1(c)), given a routing table such as the one given in Fig. 1(a), the algorithm considers all the sources one by one. For each source s, we find the most occurring port p^*, and replace all the matching rules with $(s, *, p^*)$. The remaining rules $(s, t, p \neq p^*)$ stay unchanged and have priority over the wildcard rule. Once all the sources have been considered, we do a pass over all the wildcard rules.

Flow	Output port
(0, 4)	Port-4
(0, 5)	Port-5
(0, 6)	Port-5
(1, 4)	Port-6
(1, 5)	Port-4
(1, 6)	Port-6
(2, 4)	Port-4
(2, 5)	Port-5
(2, 6)	Port-6

(a) Without Compression

Flow	Output port
(0, 5)	Port-5
(0, 6)	Port-5
(1, 4)	Port-6
(1, 6)	Port-6
(2, 5)	Port-5
(2, 6)	Port-6
$(*, *)$	Port-4

(b) With $(*, *)$ rule

Flow	Output port
$(0, 4)$	Port-4
$(1, 5)$	Port-4
$(2, 4)$	Port-4
$(2, 5)$	Port-5
$(0, *)$	Port-5
$(*, *)$	Port-6

(c) With $(n, *)$ rule

Flow	Output port
(1, 4)	Port-6
$(1, 5)$	Port-4
$(0, 6)$	Port-5
$(*, 4)$	Port-4
$(*, 5)$	Port-5
$(*, *)$	Port-6

(d) With $(*, n)$ rule

Flow	Output port
(1, 5)	Port-4
(2, 6)	Port-6
$(1, *)$	Port-6
$(*, 4)$	Port-4
$(*, *)$	Port-5

(e) Minimal solution

Fig. 1: Examples of routing tables: (a) without compression, (b) default rule only, (c) compression by the source, (d) compression by the destination, and (e) routing table with minimum number of rules.

We aggregate them using the most occurring port that becomes the default port. The default port rule has the lowest priority of all the rules.

For the second compressed routing table (Fig. 1(d)), we do the same compression considering the aggregation by destination with $(*, t, p^*)$ rules.

The third and last one (Fig. 1(b)) is the result of a single aggregation using the best default port. Choosing the smallest table between the three (aggregation by source, by destination, by default port) leads to a 3-approximation of the compression problem [17].

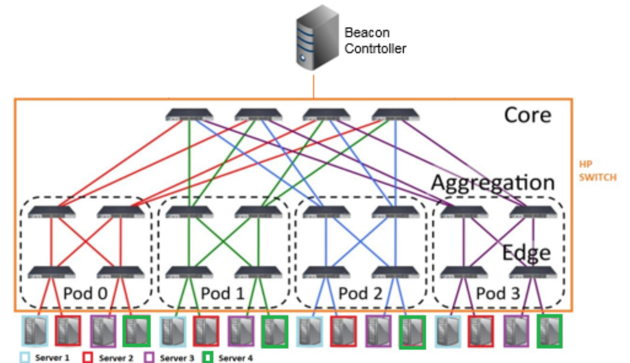

Fig. 2: Our 20 switches fat-tree architecture with 8 level 1, 8 level 2, and 4 level 3 switches.

IV. TESTBED DESCRIPTION

For the experiments, we built an experimental SDN-based data center testbed, physically composed of one HP5400zl SDN capable switch (K15.15) with 4 modules installed, each module featuring 24 GigaEthernet ports, and 4 DELL servers. All our DELL servers possess 6 multi-core processors, for a total of 24 cores, 32 GB of RAM and 12 GigaEthernet ports. On each DELL server we deployed 4 virtual machines (VMs) each with 8 virtual network interfaces all mapped to the same physical interface. Hence, we provide up to 32 different IP addresses per physical machine for total of 128 different IP addresses in the data center (or “clients”, as we will call them in the experimental part). In one of the physical servers, we deployed an additional VM that acts as the SDN controller. To avoid any problem in the network caused by a limit of resources at the controller, we configured our controller with 15 CPUs and 16 GB of RAM.

Our data center network features a $k = 4$ pods (points of delivery) full fat tree topology (see Figure 2). To enable 20 SDN switches, we configured 20 VLANs on the physical switch. Note that each VLAN belongs to an independent Openflow instance, making each VLAN an independent SDN-based switch. In every access switch we connected two VMs belonging to two different physical servers, such that all the VMs that are located in the same pod are not located on the same physical server. And each access switch hosts a single subnet. Hence, in this network architecture, there is in total 8 subnets, with 16 different IP addresses (i.e. clients) per subnet.

The HP SDN switch can support a total maximum of 65536 software rules installed on the physical switch (for all vlans together). This value is not equally distributed between the OpenFlow VLANs. It follows the concept of first flow arrived-first served. The number of hardware rules that can be stored in the TCAM memory of this switch per module being equal to 750, it was not possible to set up the 20 virtual switches of the fat-tree with this limit. Indeed, we need at least 15000 rules to represent 20 switches with 750 rules each. Therefore, we decided to use the limit reached by the total number of rules (65536). We will show that even with a small number of clients, and even with more than 750 rules per switch (3276 in average if the rules are spread equally among the switches), we will reach this limit and have some increased delay if no compression is performed.

Regarding the controller, we decided to deploy the Beacon [18] controller that will manage each switch available in the data center. According to [19], Beacon features high performance in terms of throughput, while maintaining reliability and security.

V. SETTINGS

A. Experimental scenarios

In order to assess the performance and the viability of deploying MINNIE[2] in real networks, as well as the impact that

[2]Available at: https://sites.google.com/site/nextgenerationsdndatacenters/our-project/minnie

such a solution can have on the network traffic, we ran several tests in our experimental data center under three scenarios:

1) No compression.
2) Dynamic compression with table limited size.
3) Compression when the physical switch rules' limit is reached.

In the first scenario, we fill up the routing tables of the switches and we never compress them. This scenario will provide the baseline against which we compare the cases where we execute the routing table compression in the data center. In the second scenario we set a table limit size of 1000 entries (case a) and 2000 entries (case b) on each level 2 and level 3 switches. Then we trigger the compression of a switch when it reaches its table limit size. In the last scenario, we trigger compression of all level 2 and level 3 switches when the physical switch limit (65536) is reached. This scenario illustrates the worst case where we have the highest number of rules installed in the switch and the highest number of rules to be transmitted and installed in a short period of time at compression, stressing the entire network.

B. Compressing in SDN

When the controller compresses a table, the MINNIE SDN application will first execute the routing phase and then compresses it, and once the computation finishes, a single OpenFlow command is used to remove the entire routing table. Then the new routes are sent immediately to limit the *downtime* period, that we defined as the period between the removal of all old rules and the installation of all new compressed rules. Also, when two or more switches need to be compressed at the same time, the compression is executed sequentially.

For the dynamic scenario, when a new entry must be pushed to a switch and the counter reaches the defined threshold (1000 or 2000 for cases a and b respectively), we inject first the 1000th (or 2000th) route to the switch to allow the new flow to travel to the destination, then immediately after, the compression mechanism is executed.

In our current proposition, we compress switches that are at level 2 and 3 (of the fat-tree) only, since dynamic compression at level 1 would cause inconsistencies in the network. This inconsistency is due to the fact that our compression mechanism groups both rules by destination (as it can be done classically) and by source, which can potentially lead to wrong rules for new flows. For example: suppose that we receive a packet with source s_1 going to the destination address t_1, by means of port p_1 at a given switch. We suppose also that another packet, from the same source s_1 reaches destination t_2 through port p_1. With the previous rules, if a new packet from s_1 wants to reach t_3 and does not find a matching entry in the switch's routing table, the switch will fetch the rule from the SDN controller to forward the packet. Let us suppose that the output port for this connection is p_3. However, if before having seen the packet from s_1 to t_3, for any reason, we compress the forwarding table in the switch, the entries (s_1, t_1, p_1) and (s_1, t_2, p_1) will compress to $(s_1, *, p_1)$. Hence, in this case, the packet from s_1 to t_3 will match the compressed forwarding rule and it will never reach its destination as it will be directed to port p_1 and not p_3. We thus need that the switch contacts the controller for every new connection so that our routing algorithm can take appropriate routing and load balancing decisions.

By not compressing at level 1, we make sure that any packet with an unknown destination will trigger a request to the controller. Consequently, we are able to rebuild a compressed forwarding table that does not contain inconsistent rules and we can maintain load balancing in the network.

C. Traffic pattern

In the experiments, the traffic is generated as follows: each client pings any other client in a different subnet, which means that from a single access switch, we have 16 clients pinging 112 clients. It should be noted that there were no pings between hosts belonging to the same subnet. Indeed, we wanted to focus on the compression of IP-centric forwarding rules, which is used to route packets between different subnets, and not MAC-centric forwarding rules, as the ones in legacy L2 switches.

We started our pings by transmitting 5 ping packets to a single client. We wait for this ping to terminate before sending 5 other different ping packets to another client, and so on, until all the 112 clients are pinged. When the first client finishes the execution of all those pings, a second client (hosted in the same VM) starts the same ping operation. Hence, the traffic is generated during all the period of the experiment in a round-robin manner, from client 1 to client 8 (all hosted in the same VM). Moreover, VMs do not start injecting traffic at the same time. We impose an inter-arrival period of 10 minutes between them. Hence, VM 1 starts sending traffic at minute zero, while VM 2 start at minute 10, VM 3 at minute 20, and so on. The smooth arrival of traffic in the testbed is motivated by the fact that we do not wish to overload the physical switch with openflow events. Indeed, as stated in [20], openflow commercial switches can treat up to 200 events/s. Since in our testbed we have 20 virtual forwarding elements, where each one has its own *flow_mod* (message for sending rules), *packet_out* (message with packet to be sent) and other events, the critical number of events is easily reached.

The experiment ran for almost 3.5 hours. All the rules are installed in the first 2.75 hours.

D. Number of expected rules by simulations

In our fat-tree architecture, we can easily deduce the number of rules corresponding to a valid routing for the traffic pattern mentioned above. Considering no compression at all, one rule is needed for every flow passing through a switch. The set of flows that a switch "sees" depends on its level in the fat-tree.

For any flow between two servers, the path goes through level 1 switches to which each server is connected. Every server of the n servers connected to a level 1 switch communicates with the other 7 times n servers on other subnets via outgoing and incoming flows. In total, this represents $14n^2$ flows going through any level 1 switch.

The same argument can be used to find the number of flows for switches on other levels. This represents a total of $13n^2$ flows over each level 2 switch and $12n^2$ flows over a level 3 switch. In total, $264n^2$ rules are needed throughout the entire network.

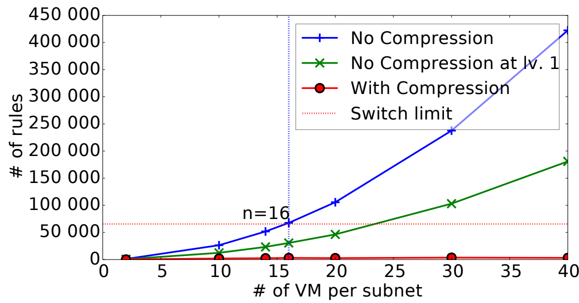

Fig. 3: Total number of rules.

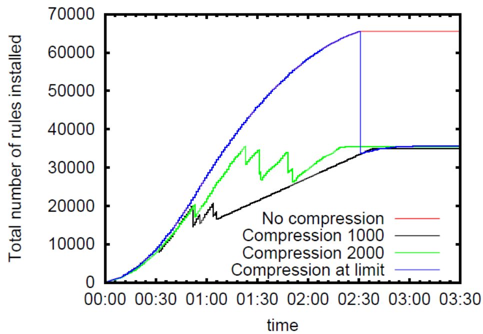

Fig. 4: Total number of rules installed on the physical switch

In Figure 3, we compare the total number of rules with no compression at all, compression on level 2 and level 3 switches only, and compression on all switches. These results are obtained via simulation. Without compression, only 15 machines per level 1 switch can be deployed without running out of space in the forwarding table of our entire data center (65536 entries), while up to 23 machines can be deployed compressing at level 2 and 3. Therefore, Figure 3 explains our choice of installing 16 clients per subnet. Indeed, it is the first value for which the number of rules exceeds our total limit of number of rules (67584 rules).

VI. RESULTS

Number of rules with/without compression. We launched the experiment with 128 clients (16 clients for each of the 8 pods). As expected and as discussed in Section V-D, when no compression is executed (Scenario 1), the limit of the switch (65536 entries) is reached. On the contrary, compressing the table using MINNIE allows to not reach the limit, be it for Scenario 2 (compressing dynamically when one of the switches reaches 1000 or 2000 entries) or Scenario 3 (compressing only when the limit is about to be reached). As shown, in Table I, the total number of installed rules does not exceed 35000 rules for both scenarios. This represents a saving of almost 50% of the total forwarding table capacity. More precisely, at level 2, we can decrease the number of rules by an order of 5 with compression at 2000 rules and by an order of 4 for compression at limit. The highest and most dramatic compression ratio is observed at level 3 switches, for which we can decrease the number of rules installed by a factor of 31 for both the dynamic compression at 2000 rules and compression at limit.

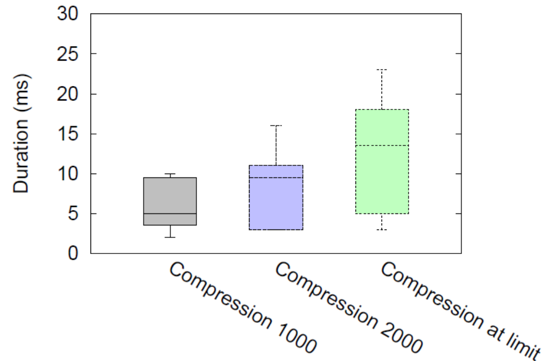

Fig. 5: Average duration of compression period.

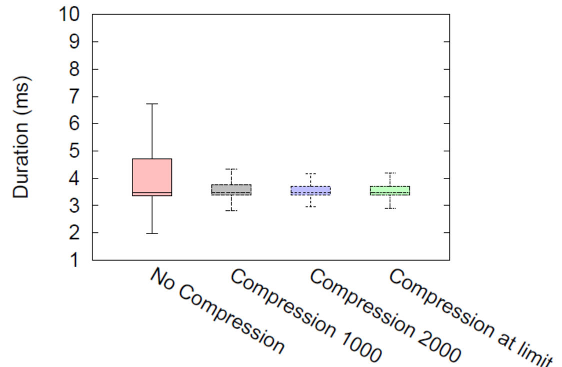

Fig. 6: Average delay of all except first packets of new flow

Figure 4 shows how the number of rules evolves over time with and without compression. First, the number of rules increases at the same pace in all 3 scenarios during the first 45 minutes. However, when compression is triggered, the number of rules decreases instantaneously. Later, in all compression cases, the number of rules increases with a lower pace since the controller will decide for some flows to use the wildcard rules, and hence, no new rules at level 2 or 3 need to be installed.

Compression time. In Figure 5, we notice that the compression duration per switch remains in the order of a few milliseconds. Indeed, the compression takes 5 ms (resp. 9 ms) in average for compression at 1000 entries (resp. 2000 entries). Even for the worst case, compression at limit (Scenario 3), it represents less than 18 ms for most of the switches. Moreover, sequentially compressing all the Level 2 and Level 3 switches requires only 152 ms. These results show that MINNIE will not introduce major problems in TCP, which is the main employed transport protocol. Since a TCP flow experiences a timeout when the RTT exceeds the smoothed RTT plus four times the RTT variation, assuming an extra delay of 5 ms (which is the average downtime period for the compression at 1000 entries), the RTT variation must be lower than 1.25 ms to avoid timeouts. This requirement is satisfied according to our experiments. Indeed, in all our experiments we observed a variance of around 40 ms and a standard deviation of 6.5 ms.

Impact of reaching the rule limit on delays. Figure 7

TABLE I: Average number of SDN rules installed in the virtual switch at each level

Level	No Compression	Compression at 1000	Compression at 2000	Compression at limit
level 1 (8 switches)	3451.625	3584	3584	3584
level 2 (8 switches)	3233.125	688.625	666.25	725.5
level 3 (4 switches)	3014.5	194	97	97.25
total (20 switches)	65536	34957	34390	34865

TABLE II: Percentage of loss

Level	No Compression	Compression at 1000	Compression at 2000	Compression at limit
%loss	0	$4.06 * 10^{-5}$	$1.36 * 10^{-4}$	$1.72 * 10^{-4}$

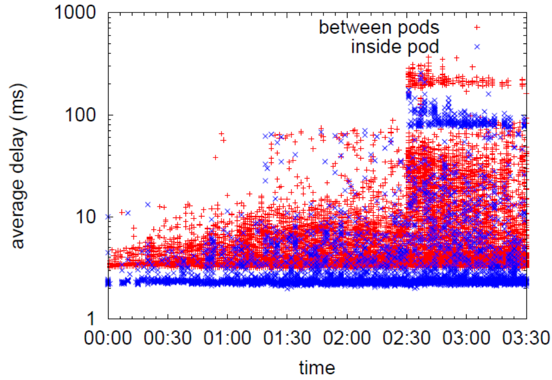

Fig. 7: Average delay: no compression, 8 clients per VM

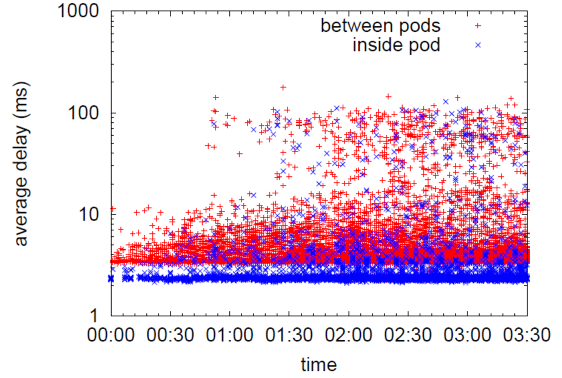

Fig. 9: Average delay: compression at 2000, 8 clients per VM

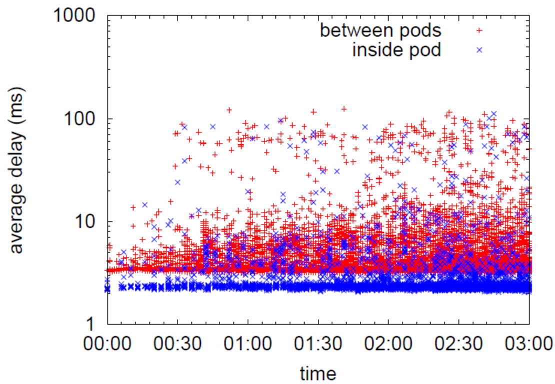

Fig. 8: Average delay: no compression, 7 clients per VM

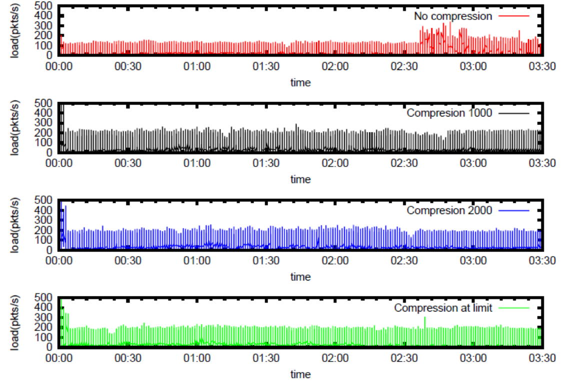

Fig. 10: Controller network traffic

shows that when we attain the switch maximum rule number, the average packet delay of some flows sharply and strongly increases (at t=2:30). Before this time, most flows experienced a delay of 1-5 ms. Afterwards, the delays of some packets jump to values over 100 ms. Indeed, at 2:30, the tables are full and at each hop, the switch on the path contacts the controller which has to forward itself the packets (rules cannot be installed). This drastically increases the average packet delay of some flows, see Figure 6. We confirm that the jumps in delay are due to the saturation of rules by comparing this experiment with n=16 to another one with n=14. In the latter case, the rule limit is not reached even without compression and no increase of delay is observed (Figure 8).

On the contrary, when using the compression algorithm (for the 16-server case), we avoid this sharp increase of delay as shown in Figure 9 for Scenario 2. Since the tables are not full, switches along the path are able to install all the news rules, and the controller is not contacted at every hop. This keeps the delay low for all compression scenarios, see Figure 6.

Impact of reaching the rule limit on first packet delay and controller load. The first packet delay provides insight on the time needed to contact the controller and install the rules when a new flow arrives. We see in Figure 11 that a typical delay for a first packet is between 10 and 20 ms. This has to be compared with the average packet delay between 1 and 5 ms (Figure 9). We observe an interesting phenomenon for the scenario without compression in Figure 12 where the first packet delay increases after 2:30. This shows that the increase

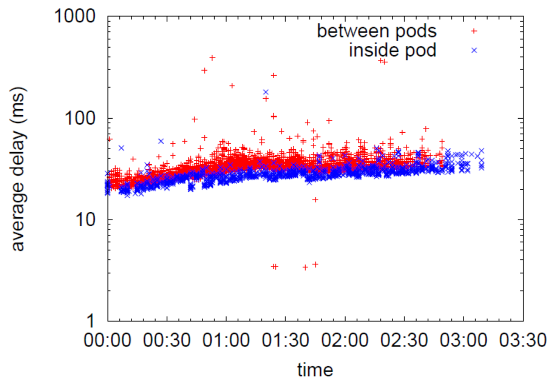

Fig. 11: First packet delay: compression at 2000, 8 clients per VM

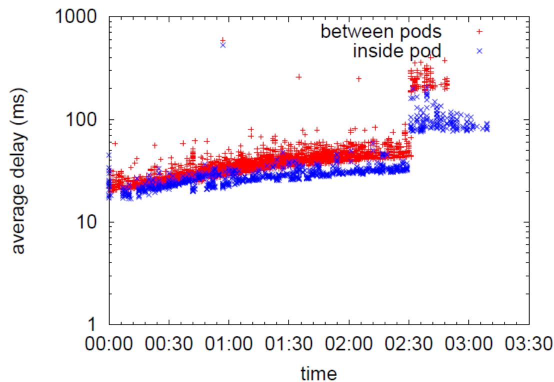

Fig. 12: First packet delay: no compression, 8 clients per VM

in average packet delay of some flows when the rule limit is exceeded has an additional explanation: it takes more time to contact the controller. Indeed, since all new flows have to contact the controller for every packet transmitted and at every hop, the load between the switch and the controller increases as shown in Figure 10. The controller starts to be saturated and takes more time to answer. On the contrary, using MINNIE allows the load of the controller to remain stable (Figure 10), except a spike of 400 pkt/sec in the 3rd scenario which corresponds to the compression of the tables of all switches when the limit is reached at t=2:40.

Impact of compression on loss rate. Last, we studied the possible impact of MINNIE on the loss rate. Indeed, when compression takes place, the algorithm first computes a new table, then deletes all the ancient rules and inserts the new rules during a short period of time. Some packets could be lost during this process. We show in Table II that the impact is negligible: loss rate $< 0.001\%$ in all compression scenarios.

VII. CONCLUSION AND FUTURE WORK

MINNIE aims at computing load-balanced routes, and at compressing SDN routing tables using aggregation by the source, the destination and with the default rule. Using experimentations, we show that our smart rule allocation can reduce drastically the number of rules needed to perform routing, with a limited impact on loss rates. Moreover, we show that when tables are full, delays increase while this behavior is not observable when compressing the routing tables. We plan to make further experimentations on the platform by considering more complex UDP and TCP traffic workload, and different topologies such as backbone networks.

REFERENCES

[1] J. Moy, "OSPF Version 2," RFC 2328 (Standard), Internet Engineering Task Force, 1998. [Online]. Available: http://www.ietf.org/rfc/rfc2328.txt

[2] "NEC univerge PF5240 and PF5820," http://www.openflow.org/wp/switch-nec/.

[3] P. Bosshart, G. Gibb, H.-S. Kim, G. Varghese, N. McKeown, M. Izzard, F. Mujica, and M. Horowitz, "Forwarding metamorphosis: Fast programmable match-action processing in hardware for SDN," in *SIGCOMM*. ACM, 2013.

[4] F. Giroire, J. Moulierac, and T. K. Phan, "Optimizing rule placement in software-defined networks for energy-aware routing," in *GLOBECOM*. IEEE, 2014.

[5] R. Cohen, L. Lewin-Eytan, J. Naor, and D. Raz, "On the effect of forwarding table size on sdn network utilization," in *INFOCOM*. IEEE, 2014.

[6] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. Gude, N. McKeown, and S. Shenker, "Rethinking Enterprise Network Control," in *IEEE/ACM Transaction in Networking*, 2009.

[7] R. Wang, D. Butnariu, and J. Rexford, "OpenFlow-based Server Load Balancing Gone Wild," in *Hot-ICE*, 2011.

[8] K. Kannan and S. Banerjee, "Compact TCAM: Flow Entry Compaction in TCAM for Power Aware SDN," in *ICDCN*, 2013.

[9] S. Banerjee and K. Kannan, "Tag-in-tag: Efficient flow table management in sdn switches," in *CNSM*, 2014.

[10] D. L. Applegate, G. Calinescu, D. S. Johnson, H. Karloff, K. Ligett, and J. Wang, "Compressing Rectilinear Pictures and Minimizing Access Control Lists," in *ACM-SIAM SODA*, 2007.

[11] C. R. Meiners, A. X. Liu, and E. Torng, "TCAM Razor: A Systematic Approach Towards Minimizing Packet Classifiers in TCAMs," in *IEEE/ACM Transaction in Networking*, 2010.

[12] ——, "Bit Weaving: A Non-prefix Approach to Compressing Packet Classifiers in TCAMs," in *IEEE/ACM Transaction in Networking*, 2012.

[13] N. Kang, Z. Liu, J. Rexford, and D. Walker, "Optimizing the "one big switch" abstraction in software-defined networks," in *CoNEXT*. ACM, 2013.

[14] Y. Kanizo, D. Hay, and I. Keslassy, "Palette: Distributing tables in software-defined networks." in *INFOCOM*. IEEE, 2013.

[15] X.-N. Nguyen, D. Saucez, C. Barakat, and T. Turletti, "OFFICER: A general Optimization Framework for OpenFlow Rule Allocation and Endpoint Policy Enforcement," in *INFOCOM*. IEEE, Apr. 2015.

[16] S. Hu, K. Chen, H. Wu, W. Bai, C. Lan, H. Wang, H. Zhao, and C. Guo, "Explicit path control in commodity data centers: Design and applications," in *Proceedings of the 12th USENIX Conference on Networked Systems Design and Implementation*, ser. NSDI'15. Berkeley, CA, USA: USENIX Association, 2015, pp. 15–28. [Online]. Available: http://dl.acm.org/citation.cfm?id=2789770.2789772

[17] F. Giroire, F. Havet, and J. Moulierac, "Compressing two-dimensional routing tables with order," in *INOC*, 2015.

[18] D. Erickson, "The beacon openflow controller," in *HotSDN*. ACM, 2013.

[19] A. Shalimov, D. Zuikov, D. Zimarina, V. Pashkov, and R. Smeliansky, "Advanced study of SDN/openflow controllers," in *CEE-SECR*. ACM, 2013.

[20] D. Kreutz, F. Ramos, P. Esteves Verissimo, C. Esteve Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proceedings of the IEEE*, vol. 103, no. 1, Jan 2015.