Performance Evaluation of Flow Creation Inside an OpenFlow Network

Walber José Adriano Silva

***Abstract*— A top concern in Software Defined Networking (SDN) is the management of network flows because of resource limitation in SDN devices (such as TCAM size) and the signaling overhead between the control and data plane elements. This work performs an evaluation of reactive, proactive and active strategies for creating flows inside an SDN network. The results indicated a trade-off between memory space utilization and signaling overhead, where each approach has benefits and drawbacks. The proactive approach is more suitable when all paths are predicted with high accuracy, resulting in high memory consumption and low signaling overhead. When the network requires frequent adjustments, the reactive strategy is indicated providing a low memory consumption and high signaling intensive approach. Finally, the active strategy can be used when memory and signaling are critical problems because of the median memory consumption and signaling overhead.**

***Keywords*— OpenFlow Network; Software Defined Networking; Performance Evaluation**

I. Introduction

Software Defined Networking (SDN) is emerging as new network paradigm that separates software and hardware roles in devices (vertical integration), generalizes network devices and functions, enables programmability of networks, and centralizes network management tasks[1]. The major feature of SDN is the decoupling between control and data planes. The centralization logic of SDN controller can allow a consistent view of the network state, bring programmability to the network, and also enforce network policies, routing decisions and forwarding information.

There is some instantiation of the concepts of SDN [2], and one of them is OpenFlow protocol [3], which is becoming a standard *de facto* instance of SDN on the academic and industry field [4]. In an OpenFlow network, the OpenFlow switches often are deployed with Ternary Content Addressable Memory (TCAM). TCAM memories are very fast, but expensive, require much space in the chip of an OpenFlow switch and have a high energy consumption.

A challenging network management task is the creation of an efficient quantity of flow inside an OpenFlow network due to the limitation of the resources (e.g. switches memory) and the signaling overhead between data and control plane elements (OpenFlow switches and controllers, respectively) [5]. This way, if incoming packets do not found a match pattern in the OpenFlow switch table (a table miss event), usually, the default action is to contact the OpenFlow controller to provide instructions. Because the quantity of the signaling to create new flows is proportional to the size of the network, the overall number of control packets between controller and switches may become problematic.

Center of Informatics, Federal University of Pernambuco, Pernambuco-PE, Brazil, E-mail: wjas@cin.ufpe.br

For OpenFlow networks, it is an issue to decide where and when a network flow rule has to be installed. Therefore, the aim of this paper is to evaluate the performance of flow creation inside an OpenFlow Network. The main contributions of this work are the follows:

1) Description and analysis of three algorithms for flow creation inside an OpenFlow Network: reactive, proactive and active;
2) Performance evaluation of the three algorithms with an emulation environment;
3) A discussion about the trade-off among OpenFlow rule installation and the overhead communication between OpenFlow switches and OpenFlow controller.

The remain of this work is organized as: Section II describes the relevant works in the field of rule placement problem; Section III details the three algorithms to be analyzed; Section IV presents the experiments and description of the results; Section V discusses the trade-off between signaling and memory utilization in OpenFlow networks; Section VI highlights the conclusions of this work.

II. Related Works

With regard to reducing the number of rules installed into the SDN network, the works of *Palette* [6] and *One Big Switch* [7] are proposals that tried to addressed the OpenFlow switches memory limitation. They considered that the rules to be installed are non-reducible, so they can not enforce rule aggregations. Thus, the solutions distributed the routing rules in the network in such a way that the routing semantics are maintained, and the network policies are not violated.

The work of [8] also seeks to optimize the placement of routing rules within an SDN network. It accomplishes this by minimizing the resources required for the treatment of network flows. With an algebraic model and using the Integer Linear Programming, an optimization technique, to express constraints in the end-to-end routing policy on the network, the work indicated how to allocate a greater amount of traffic over memory capacity constraints using the model proposed. The work also performs comparisons with the solutions *Palette* and *One Big Switch* and found similar values of optimization. However, it overcomes the previous works in a scenario of extreme memory shortage, when the SDN controller must be used to maintain the minimum network operating state, even if the network performance degradation occurs.

Furthermore, to reduce signaling overhead between OpenFlow switches and OpenFlow controllers, it is necessary a

prediction, or estimation of the network traffic, to allow rules installation in advance, and before the traffic ingress into the network. The accurate achievement prediction requires data collections and induces signaling messages, which makes it a difficult task.

For increase the utilization of TCAM space and avoid TCAM misses, the authors in [9], presented a system that combines an adaptive heuristic with proactive eviction by choosing the timeout values of OpenFlow rules. They found that over particular types of network, with the understanding of the network traffic, is possible to outperform static timeout policy (fix value of OpenFlow timeout rules). However, to increase the utilization of the data plane elements memory, it requires a heavy signaling overhead to the controller, because of frequently fetching information about the switches and flows states.

Thus, a trade-off among rule installation and the signaling overhead between OpenFlow switches and OpenFlow controller must be made. The work in [5] classified the flow creation for OpenFlow networks into two categories: reactive, the rules are created on demand to react upon flow events; proactive, rules are populated in advance, that way the flow is created before the packet arrives at an OpenFlow switch port.

Furthermore, the work of [10] already has compared reactive and proactive approaches. However, one other option to create flows inside an OpenFlow network is to use the bird's eye of SDN to active create network rules when the controller already discovers the path that the packet will take. Therefore, no previous works performed evaluation of the three groups of flow creation (reactive, proactive and active), with regard to memory capacity and signaling. The following section presents the descriptions of this three approaches.

III. Flow creation

Before delving into the algorithms, some concepts need to be defined. An OpenFlow network uses the concept of flow to carry traffic inside the network. A flow is a sequence of packets sent from a particular source s to a particular destination d following a given path [11], where the packets match with the same fields values of a flow entry.

A path is an ordered sequence of OpenFlow switches and links from a given origin to the destination. Another concept is the endpoint. They are places inside an SDN network where the network operator has some interest for packets to reach that location. An endpoint (element belong to set O) can be a host, a switch port connected to a load balancer appliance, the ingress/egress of the backbone network, and others. Thus, an endpoint consists of two elements. The first one is the source, where the packets are matching to belong a specific flow, and the destination, the place where the packets are released from the flow. Specifically, this work models an OpenFlow network as a directed graph (or digraph) $G(V, E)$, where V is a set of nodes (e.g. OpenFlow switches) and E a set of edges (e.g. network links).

The topology of the OpenFlow network is assumed to be known for applying the flow creation algorithms. Thus, the OpenFlow controller has an instance of the data structure of G. One way to reach that topology information is fetching OpenFlow switches with control packets to discover the origin and destiny of the links, switches, and hosts. Once the topology is known, the problem is to discover the benefits and drawbacks of three algorithms for rule creation inside an OpenFlow network. The three algorithms to evaluate are: reactive flow creation; proactive flow creation; active flow creation.

Algorithm 1 Reactive flow creation

Input: opi, G

1: $s \leftarrow get\_src(opi, G)$
2: $d \leftarrow get\_dst(opi, G)$
3: $v \leftarrow get\_switch(opi, G)$
4: $action \leftarrow get\_next\_hop(d, v)$
5: **if** $FT(v, t) < C_v$ **then**
6: $out\_port \leftarrow SW(v, next\_hop)$
7: $match \leftarrow MT(s, d)$
8: $opo \leftarrow create\_openflow(out\_port, match, action)$
9: $send(opo, v)$
10: **end if**

A. Reactive creation

The Reactive flow creation populated the flows on demand to react upon incoming packet events, OFPT_PACKET_IN messages[4]. When a packet that did not match any rule installed into an OpenFlow Switch, often, the switch enqueues the packet and informs the controller for a new flow creation. Afterward, the controller computes the rules to be associated with the new flow and installs them in the network. Once the rules are installed, on the switches, packets are dequeued and forwarded in the network. The freshly installed rules will then process any subsequent packet of the flow without further intervention of the controller [5].

Algorithm 1 describes how the SDN controller creates the flows following the reactive creation logic. The algorithm extracts from the Packet-In event (*opi*), the information about which OpenFlow switch originated the event, the source and destination for the new flow (lines 1- 3). In line 4, the function $get\_next\_hop(\bullet, \bullet)$ receives the destination and the identification of the OpenFlow switch, and returns the $action$ information, which contains the output port and message to modify the flow table. In line 5, it is verified whether the OpenFlow switch has the capacity to install a new flow, then an *opo* OpenFlow OFPT_FLOW_MOD packet[4] is sent to the OpenFlow switch to install the rule with matching, action and output port information for new flow constructed (lines 6-9).

B. Proactive flow creation

The algorithm to create a new flow with proactive logic is present in Algorithm 2. Proactive flow creation uses the network topology information of an OpenFlow network (G) and the set of all endpoints (O). Thus, before a packet, belonging to a particular flow, arrives at the OpenFlow network, all flow rules have already been installed into the OpenFlow switches for that packet.

Algorithm 2 Proactive flow creation

Input: G,O

1: **for** (s,d) in O **do**
2: $path \leftarrow shortest\_path(s,d)$
3: $inst \leftarrow create\_openflow\_instructions(path)$
4: **for** v in $path$ **do**
5: **if** $FT(v,t) < C_v$ **then**
6: **if** d is connected to v **then**
7: $out\_port \leftarrow inst[v,d]$
8: **else**
9: $out\_port \leftarrow inst[v,v+1]$ {v+1 is the next hop}
10: **end if**
11: $action \leftarrow forward\_to(d, out\_port)$
12: $match \leftarrow MT(s,d)$
13: $opo \leftarrow create\_openflow(out\_port, match, action)$
14: $send(opo, v)$
15: **end if**
16: **end for**
17: **end for**

Algorithm 3 Active flow creation

Input: opi,G,O

1: $s \leftarrow get\_src(opi, G)$
2: $d \leftarrow get\_dst(opi, G)$
3: **if** (s,d) in O **then**
4: $path \leftarrow shortest\_path(s,d)$
5: $inst \leftarrow create\_openflow\_instructions(path)$
6: **for** v in $path$ **do**
7: **if** $FT(v,t) < C_v$ **then**
8: **if** d is connected to v **then**
9: $out\_port \leftarrow inst[v,d]$
10: **else**
11: $out\_port \leftarrow inst[v,v+1]$ {v+1 is the next hop}
12: **end if**
13: $action \leftarrow forward\_to(d, out\_port)$
14: $match \leftarrow MT(s,d)$
15: $opo \leftarrow create\_openflow(out\_port, match, action)$
16: $send(opo, v)$
17: **end if**
18: **end for**
19: **end if**

The algorithm is executed during initialization of the network. It creates OpenFlow rules for each pair source and destination (s,d) of the set of endpoints O (lines 1-17) using a $shortest\_path(\bullet,\bullet)$ function, that return the shortest path between s and d (line 2). Afterward, a dictionary of instructions is generated to configure the flows for each OpenFlow switch (line 3) with the function $create\_openflow\_instructions(\bullet)$. Once the controller has the information about the path and the set of instructions, it sends OpenFlow messages for every OpenFlow switch of the path (lines 4-16).

In line 5, it is verified whether the OpenFlow switch has the capacity to install a new flow. Then it is also checked if d is connected to the current selected switch v (line 6). If it is the case, the $inst$ dictionary returns the port where d is connected, otherwise the port of the next hop of the path. With that all the information for create a flow, an opo OpenFlow OFPT_FLOW_MOD packet is sent to the OpenFlow switch v to install the rule with the match pattern, action and output port information for a new flow (lines 12-14). The process of flow creation continues until all the endpoints have rules installed into the OpenFlow network.

C. Active flow creation

The algorithm for active flow creation has similarities with the reactive and proactive flow creation. The difference between active and reactive is that in active algorithm all OpenFlow Switches receive OpenFlow rules at once, instead of the reactive algorithm which requires each OpenFlow Switch, individually, communicate with the controller to create the new flow. Also, it is important to highlight the difference between active and proactive. Active does not create rules in advance. It waits for the packet from source to destination arrive into one OpenFlow Switch. Then, the switch sent an OpenFlow message to the controller that process and compute the path from source to destination for that particular packet. Afterward, the controller sent to all switches in the path new OpenFlow messages instructing them to create the new flow. Therefore, the active flow creation algorithm is a mix of reactive and proactive approaches.

Algorithm 3 describes the active flow creation. The lines 1 and 2 extract the information from source s and destination d of the opi, OpenFlow packet input event. This information is used to discover if the pair (s,d) is one of the endpoints inside the set O. Afterwards, lines 4-18 are used to create the flow for each OpenFlow switch in the path between source and destination, in the same way of the proactive flow creation.

D. Numerical analysis

This subsection analysis the aforementioned algorithms. In the first moment, we derive an analytic model to calculate the number of update messages (signaling packets). Subsequently, it is presented the numerical evaluation of the memory utilization.

The reactive creation requires the OpenFlow switch to send and receive control information to the SDN controller. Because of it, the number of signaling messages is $2n$, where n is the number of the next hop element in network flow path $P(s,d)$. To the active flow creation, the first packet of the flow is sent to the controller. The controller decides the set of OpenFlow switches that belong to the flow path between source and destination of the packet. Then, each switch in this path receives one control packet for creation of the new flow. Considering the first message received by the controller, the exact number of signaling messages are $n+1$, where n is the number of the switches in the path $P(s,d)$. Finally, for proactive strategy all rules are created in advance for each OpenFlow switch, which requires n signaling messages.

With regard to analytical modeling, the OpenFlow switch memory required to install the flows inside the OpenFlow

network, the memory utilization is proportional to the number of flows in a specific moment in time of the network for reactive and active algorithms. And the quantity of flows is related to the number of endpoints in a particular moment of time, or $O(t)$. However, for the proactive approach, it creates all possible flows in advance, the utilization of memory has a fix consumption that depends on the number of endpoints to be connected, or O (set of endpoints). Therefore, for reactive and active algorithms requires at most the same memory utilization of the proactive algorithm, $||O(t)|| \leq ||O||$.

Considering d the number of default OpenFlow rules required to operate the OpenFlow network (e.g. rules to send the packet to the controller for treatment of a missing matching event), v number of switches in the OpenFlow topology, and $path(\bullet, \bullet)$ a function that generates a path from source to destination. The generalization of the memory utilization for asymmetric traffic network paths is given by:

$$U(t) = d * v + \sum_{(src,dst)}^{O(t)} (||path(src, dst)|| + ||path(dst, src)||)$$

When the network traffic follows symmetric paths inside an OpenFlow network, or in other words, all packets from source to destination also travel back through the same path, the memory utilization can be simplified to:

$$U(t) = d * v + 2 \sum_{(src,dst)}^{O(t)} ||path(src, dst)||$$

IV. Performance Evaluation

A. Virtual Environment

To investigate the three flow creation algorithms, two computers were used. The first computer was a dedicated machine with Ubuntu 16.04 LTS, 8 GB of RAM and 8 CPU core with a clock of 2.20 GHz, and it was used to be the controller. The second computer has a 3 GB of RAM machine, with CPU clock of 2.4 GHz and Ubuntu version 14.04.4 LTS, and it executed the Mininet [12] for the network emulation environment. A physical Ethernet cable of 100Mbps in crossover mode connected the two computers.

B. Case study - Abilene topology

A realistic topology is adopted to evaluate the three strategies. The topology was extracted from Internet Topology Zoo[1] dataset [13]. The 2005' Abilene topology was selected to apply the experiments. Figure 1 depicts the Abilene topology representation. This topology has 11 nodes and 14 links and was reproduced inside the Mininet emulation environment.

The 2005' Abilene topology was used to investigated the effect of signaling overhead for reactive and active approaches. The metrics *trip time* and *memory utilization* were investigated based on the percentage of endpoints connected in the network.

[1]An ongoing project that collects network topologies information from around the world.

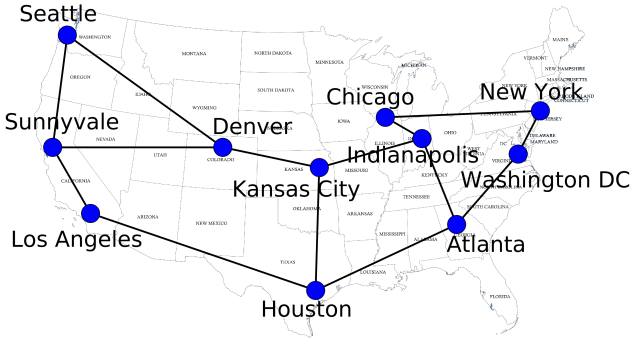

Fig. 1. The representation of Abilene topology.

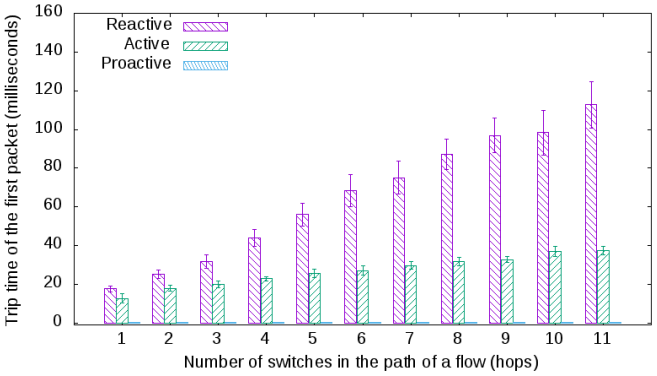

Fig. 2. The trip time of the first packet to create OpenFlow rules.

1) Trip time: it is defined as the time for the first packet travels the path from a source to the destination, and also includes the time for creation OpenFlow rules in every switches that belongs to that path. The goal here is to analyze the impact of signaling overhead into the trip time of the first packet that creates a flow. For this purpose, the experiments used Internet Control Message Protocol (ICMP) packets from an endpoint (a host source) to another endpoint (a host destination). To avoid additional latency during the experiments, caused by Address Resolution Protocol (ARP) resolution, we guaranteed that hosts already had discovered all other hosts, and then they did have information about the MAC addresses of each other.

The experiments consisted of making one host to send an ICMP packet to another host n hops away. The longest path size in Abilene topology is $n = 11$ hops, where one OpenFlow switch is visited at least once. After the measurement of the trip time was done, the experiment waits for the timeout of the flow to be reachable and repeats the process. For the three algorithms, it was collected more than 30 times the value of the trip time between endpoints, computed the average trip time, and applied the confidence interval with 95%. The results are presented in Figure 2.

2) Memory utilization: It was randomly selected a certain percentage of the available hosts (endpoints) and measured the number of rules installed into the OpenFlow Switches. Because the number of endpoints depends on a particular time of the network, the experiments were repeated 300 times for each percentage of endpoints in use. Afterward, it was measured the average of the number of rules installed and

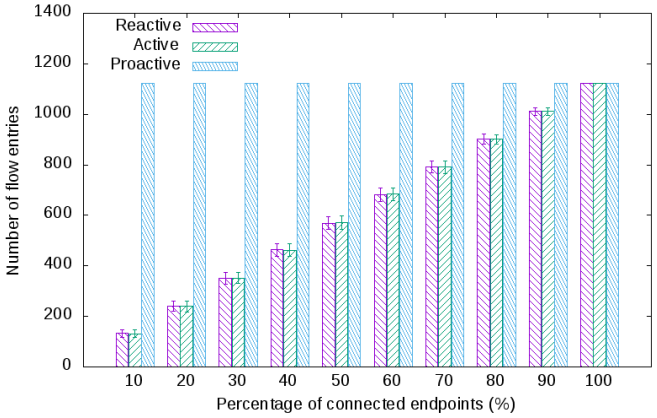

Fig. 3. Total number of configured flow entries.

estimated the confidence interval with 95%. The results are plotted in Figure 3. The reactive and active strategies are statistical equivalent when considering the percentage of endpoints connected, and them are only equals to proactive strategy when all (100%) endpoints are required to be connected.

V. Discussion

Considering the signaling requirements to install the OpenFlow rules, Figure 2 indicates that reactive and active algorithms increase the latency for the first packet of a flow. The reactive algorithm is more critical than the active approach when the number of hops crossed by the packet increases because the reactive algorithm uses the controller for flow creation intensively, as indicated and predicted by our numerical analysis of the number of signaling messages. Furthermore, the proactive algorithm keeps the trip time of the first packet minimal, because when a packet arrived into one of the OpenFlow Switches, it has already installed the rules for that traffic flow.

With regard to memory utilization, Figure 3 presents the results for the number of flows entries used for a given percentage of connected hosts in the topology adopted. As expected, the proactive algorithm requires high values of switches memory utilization to keep the endpoints full connected, even if there are no network traffic for a given rule. Hence, proactive strategy can not be adopted to connect all endpoints when a network operator is dealing with a medium or large sized network, because it is not feasible to install all possible flows between endpoints inside the data plane.

One drawback of the reactive and active flow creation algorithms is the duration of time the rule has to be inside an OpenFlow switch. To avoid rules that are no longer useful, the authors of [9] investigated the values for the timeout in an OpenFlow network, and they suggested that a dynamic policy has to be adopted to choose the proper timeout value. This decision has to include the current memory utilization, flow state, network traffic, controller capacity and data plane element location [9], [14], [15]. The memory limitation of the OpenFlow switches impose a network management challenge, and the memory consumption for reactive and active strategies depends on the number of elements that are being connected in the network for a given instant of time.

VI. Conclusion

Based on the achieved results, the proactive solution is more indicated for the treatment of flows that requires low latency (for example, real time traffic), for a few number of endpoints (memory limitation) or when the traffic flow is previously known. The reactive and active algorithms reduce the utilization of memory when compared with the proactive approach, since they can dynamically adapt to network traffic, but are signaling intensive, which could consume a lot of the SDN controller resources (e.g. CPU). Moreover, the active algorithm requires less time for flow creation than the reactive approach.

Acknowledgment

Professors Djamel Fawzi Hadj Sadok and Kelvin Lopes Dias for the support during the development of this work.

References

[1] B. A. A. Nunes, M. Mendonca, X. N. Nguyen, K. Obraczka, and T. Turletti, "A survey of software-defined networking: Past, present, and future of programmable networks," *IEEE Communications Surveys and Tutorials*, vol. 16, no. 3, pp. 1617–1634, 2014.

[2] N. Feamster, J. Rexford, and E. Zegura, "The road to SDN: An Intellectual History of Programmable Networks," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 2, pp. 87–98, 2014.

[3] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: Enabling Innovation in Campus Networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, p. 69, 2008.

[4] B. Pfaff, B. Lantz, B. Heller, C. Barker, D. Cohn, and M. Casado, "OpenFlow Switch Specification - 1.3 version," *Open Networking Foundation*, vol. 0, pp. 0–105, 2012.

[5] X.-n. Nguyen, D. Saucez, C. Barakat, and T. Turletti, "Rules Placement Problem in OpenFlow Networks: A Survey," vol. 18, no. 2, pp. 1273–1286, 2016.

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

[7] N. Kang, Z. Liu, J. Rexford, and D. Walker, "Optimizing the "one big switch" abstraction in software-defined networks," *Proceedings of the ninth ACM conference on Emerging networking experiments and technologies - CoNEXT '13*, pp. 13–24, 2013.

[8] X.-n. Nguyen, D. Saucez, C. Barakat, T. Turletti, I. Sophia, and A. Méditerranée, "Optimizing Rules Placement in OpenFlow Networks : Trading Routing for Better Efficiency," *HotSDN 2014*, no. HotSDN, pp. 127–132, 2014.

[9] A. Vishnoi, R. Poddar, V. Mann, and S. Bhattacharya, "Effective Switch Memory Management in OpenFlow Networks," *8th ACM International Conference on Distributed Event-Based Systems*, pp. 177–188, 2014.

[10] M. P. Fernandez, "Comparing OpenFlow controller paradigms scalability: Reactive and proactive," *International Conference on Advanced Information Networking and Applications, AINA*, pp. 1009–1016, 2013.

[11] D. Kreutz, F. M. V. Ramos, P. E. Veríssimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-Defined Networking : A Comprehensive Survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14 – 76, 2015.

[12] Mininet, "Mininet - An Instant Virtual Network on your Laptop (or other PC)," 2016. [Online]. Available: http://mininet.org/

[13] Topology-zoo, "The internet topology zoo," 2017. [Online]. Available: http://www.topology-zoo.org/dataset.html

[14] X. Wang, C. Wang, J. Zhang, M. C. Zhou, and C. Jiang, "Improved Rule Installation for Real-Time Query Service in Software-Defined Internet of Vehicles," *IEEE Transactions on Intelligent Transportation Systems*, 2016.

[15] W. J. A. Silva, K. L. Dias, and D. F. H. Sadok, "A Performance Evaluation of Software Defined Networking Load Balancers Implementations," in *International Conference on Information Networking (ICOIN)*, 2017.