Noname manuscript No.
(will be inserted by the editor)

ARP-P4: Deep analysis of a hybrid SDN ARP-Path/P4-Runtime switch

Isaias Martinez-Yelmo · Joaquin Alvarez-Horcajo · Miguel Briso-Montiano · Diego Lopez-Pajares · Elisa Rojas ·

Received: date / Accepted: date

Abstract The Software-Defined Networking (SDN) architecture decouples the control plane from the data plane, but it does not explicitly state where the control should be located. This article analyses the benefits of maintaining the control as close as possible to the data plane, instead of the more traditional centralised control plane approach. To this purpose, it delves into the study of ARP-P4, a hybrid software switch defined by using the P4 language to facilitate its future use and deployment in P4 targets. Its hybrid properties come from supporting two complementary different ways of establishing paths: a centralised SDN approach based on P4-Runtime and a traditional distributed approach based on the ARP-Path protocol that obtains a similar performance to centralised solutions based on Equal Cost Multi-Path (ECMP) and Dijkstra. The results show the feasibility of hybrid devices that combine different forwarding paradigms without losing performance with respect to well-known solutions such as ECMP, and how their combined use can lead to enhance and scale communication networks.

Keywords P4 · P4-Runtime · SDN · Hybrid SDN

Isaias Martinez-Yelmo
Automatics Department. University of Alcala. 28805, Alcala de Henares, Spain
ORCiD: 0000-0001-9648-8669
Tel.: +34 91 885 68 31
E-mail: isaias.martinezy@uah.es

Joaquin Alvarez-Horcajo
Automatics Department. University of Alcala. 28805, Alcala de Henares, Spain
ORCiD: 0000-0002-8522-9933

Miguel Briso-Montiano
GMV Inc., 28760, Tres Cantos, Madrid, Spain.

Diego Lopez-Pajares
Automatics Department. University of Alcala. 28805, Alcala de Henares, Spain
ORCiD: 0000-0002-8959-4321

Elisa Rojas
Automatics Department. University of Alcala. 28805, Alcala de Henares, Spain
ORCiD: 0000-0002-6385-2628

Acknowledgements This work was funded by grants from Comunidad de Madrid through project TAPIR-CM (S2018 / TCS-4496), and by the University of Alcala through the grants "Formacion del Profesorado Universitario (FPU)" and "Ayuda de Iniciacin en la Actividad Investigadora" and the project CCG2018-EXP / 076.

1 Introduction

Due to the current trends of softwarisation and virtualisation, enhanced networking devices are emerging with extended programmability and Software-Defined Networking (SDN) support. However, although the SDN paradigm is pivotal towards next generation communication networks [1], its founder protocol, OpenFlow [2], is unable to cope with the strict demands of new stakeholders [3], such as data plane programmability. A novel approach to overcome this limitation is the ARP-P4 switch [4] since it follows a hybrid approach. The ARP-P4 switch combines both SDN and traditional standalone architectures in a single device. The simultaneous support of both approaches provides new insights and features as an alternative to the dummy-switch approach of the traditional SDN architecture. Indeed, some authors advocate for still conveying part of the intelligence of the network at the SDN data plane to guarantee further possibilities. This concept is the so-called hybrid approach [5], and some authors have already proved its benefits [6, 7]. This paper studies ARP-P4 [4] in deep, a hybrid SDN ARP-Path/P4-Runtime switch based on the P4 language [8] and the P4-Runtime specification [9]. These technologies are initiatives that aim to fulfil the limitations of OpenFlow by allowing the programmability of device data planes. Thus, they are suitable solutions for the definition of new featured devices such as ARP-P4. Furthermore, ARP-P4 data plane capabilities may be extended in the future since it is based on the P4 language.

The main contributions of the paper are the following ones. (1) A deep study of the autonomous, P4-Runtime and hybrid capabilities and performance of ARP-P4- devices. Although the results are promising, some limitations exist. Some limitations are due to the current P4 language specification, which is used for the ARP-P4 behaviour definition, since P4 has been developed by mainly considering non-autonomous SDN devices. Other limitations are related with the existing constraints on how the target platforms support the different P4 functionalities. Therefore, (2) the problems of the Behavioral Model v2 (BMv2) target are investigated and exposed thoroughly. Moreover, this work demonstrates how P4 can be used to define and deploy hybrid SDN/traditional networks and devices, which enhances the possibilities of future communication networks. Finally, (3) a functional study on the hybrid ARP-Path/P4-Runtime SDN capabilities of ARP-P4 devices. The hybridisation enables to ARP-P4 devices the acquisition of additional capabilities such as the on-demand configuration of access lists (ACLs) via P4-Runtime but maintaining the forwarding decisions on the data plane.

The rest of the paper is organised as follows: section 2 describes the related work associated with ARP-P4. Section 3 explains the design decisions and how ARP-P4 has been developed and section 4 show the results from our evaluation of ARP-P4. Finally, section 5 summarises the detected problems and obstacles during the development of ARP-P4, and section 6 collects the conclusions from this work. Related future work research lines are highlighted in section 7.

2 Related Work

Despite SDN is becoming the key technology for next generation communications networks [1], it still presents some disadvantages that must be addressed. Thus, there are proposals that maintain certain control capabilities in the data plane to leverage the responsibility on SDN controllers, which decreases the dependability of the data networks on the control plane [6]. Some previous related work exists such as [5, 7], but they have a limited impact since it is a softwarised proof-of-concept. An alternative solution with a broader impact would be desirable .

The P4 language [8] is part of the P4 Language Consortium, which recently joined the Open Networking Foundation (ONF) and Linux Foundation. This P4 language is getting the focus of the networking research community. It is a high-level language designed to provide fine-tuned and unambiguous programmability of data planes. P4 defines all actions since the moment a packet is matched until it is forwarded, processed or discarded. P4 is compiled against specific *targets* that support it. Additionally, P4 can make use of extern objects that are architecture (target) specific constructs. These extern objects are used by P4 programs through well-defined APIs but their internal behaviour is hard-wired and dependent on the target; hence, they are not programmable using P4. Some of them are standardised by P4 such as the Packet Replication Engine (PRE), which is an extern that configures multicast groups by copying packets to the required egress ports. Moreover, P4 is complemented by *P4-Runtime* [9]. While P4 defines the data plane prior to deployment, P4-Runtime communicates this data plane with SDN controllers to provide runtime capabilities. Hence, both P4 and P4-Runtime aim to substitute and improve *traditional* SDN data planes and the OpenFlow protocol, respectively. For instance, the Open Network Operating System (ONOS) platform [10], implemented by the ONF community, features one of the most advanced SDN controllers supporting P4. Although the P4 definition takes into account the SDN architecture, it does not imply that cannot be used to define autonomous capabilities in a data plane. Indeed, ARP-P4 [4] shows how autonomous capabilities can be defined in a P4 based data plane. More specifically, ARP-P4 supports the ARP-Path protocol to establish paths autonomously in a layer 2 domain. Unfortunately, it has been only tested in the BMv2 [11] software target. Thus, performance is not its main goal. Its functionality is deployed via JSON files (obtained from the compilation of P4 code) dynamically at runtime.

Finally, it is important to remark the capabilities of ARP-Path in order to properly understand how ARP-P4 can forward packets autonomously. ARP-Path [12] is a shortest-path exploration protocol for switches. Contrarily to Shortest Path Bridging (SPB) [13] and TRILL Routing Bridges (RBridges) [14], which compute paths based on link-state information, ARP-Path leverages ARP Request frames (broadcast frames) to explore the network and find the shortest path (sink tree) for a given source. Moreover, ARP-Path implements a simple *lock* mechanism (instead of the traditional Spanning Tree Protocol) that prevents temporarily relearning and flooding a source MAC address just previously associated with an ingress port to avoid loops. Thus, it discards late copies of the same frame that may arrive at other different ingress ports. ARP-Path is an efficient protocol that not only discovers paths with a single probe frame, but it also reduces the computation to calculate them and provides minimum-latency paths at the time that they are created. This property is because path creation in ARP-Path considers the current status of the network and, hence, paths are able to avoid bottlenecks or heavily loaded links, which SPB and RBridges might totally ignore since their paths are statically calculated based on fixed costs.

3 ARP-P4

This paper aims to study the performance, behaviour and functionality of ARP-P4. A key aspect is how ARP-P4 is able to support its ARP-Path/P4-Runtime hybrid functionality and the autonomous capacity that can obtain without hindering the decoupling principles of the SDN paradigm.

As it was stated previously in section 2, ARP-P4 is defined by using the P4 language that allows the programmability of ad-hoc data planes. Moreover, it is necessary a target with P4 support. The only current supported target is BMv2 [11, 15] because it is the one designed for prototyping. Furthermore, limitations of the P4 language, related to its assumption of the existence of an external SDN control plane, are a non-negligible constraint as it is explained in section 5.

3.1 High-level design

ARP-P4 is a hybrid SDN ARP-Path/P4-Runtime device that can establish and manage paths and forward packets by matching of rules inserted by an external controller with P4-Runtime support or autonomously by using the ARP-Path protocol if no P4-Runtime rules apply for an incoming packet. Thus, an ARP-P4 device follows the defined steps depicted in Fig. 1. When an ingress packet arrives, an ARP-P4 device first checks if any P4-Runtime matching entry exists. If a match exists, the defined action with the matching entry is applied to the ingress packet. Later, on the one hand, if the egress port is set up by the P4-Runtime rule, the packet processing is finished. On the other hand, if the egress port is not set up (i.e. an ACcess List (ACL) rule), ARP-P4 checks if an ARP-Path matching entry is applicable. Moreover, if a P4-Runtime matching entry does not exist, an ARP-P4 device checks if an ARP-Path matching entry is applicable since an egress port

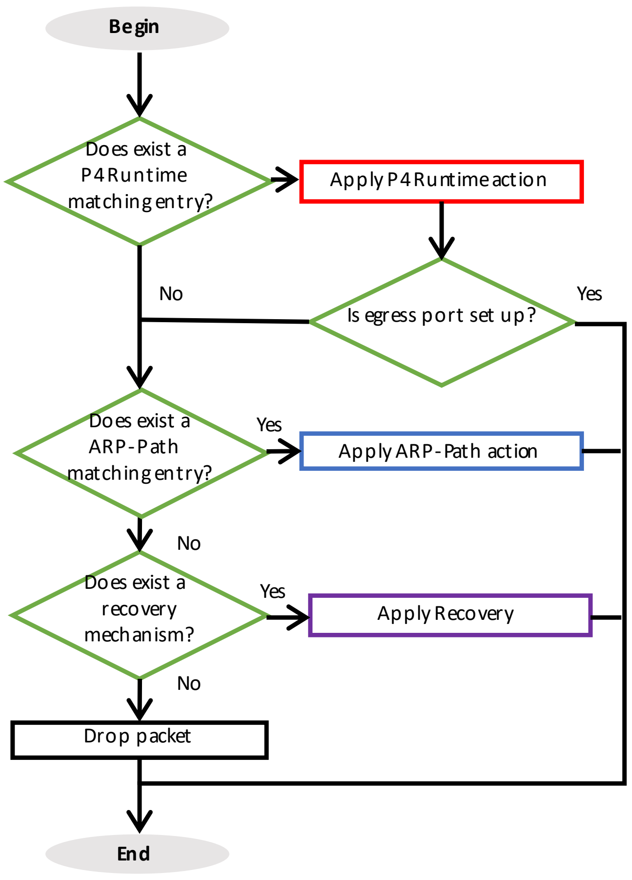


Figure 1 ARP-P4 processing and forwarding of packets

is necessary. It could occur that ARP-Path may not also have a matching entry. In that case, unless an optional recovery mechanism exists, the packet is dropped. The behaviour is simple, but it is also necessary to take into account how to define and manage the data structures to support the matching entries and their rules.

The P4-Runtime matching entries are easily defined in the P4 language [8] by using P4 tables that allow the association of a matching entry with a set of rules. On the one hand, the P4 language has the flexibility to easily define the required fields of ingress packets that will be used to define the desired matching entry. On the other hand, there is a set of existing actions that allow the definition of the required set of rules to process packets in a P4 device. However, the definition of ARP-Path in the P4 language is not simple since it is not specifically designed to support an autonomous data plane. The requirements to support ARP-Path by ARP-P4 are as follows. ARP-Path needs a special table, the Learning Table (LT), where ARP-Path keeps the different learned MAC Address like a legacy switch and later this table is later used to forward packets with the previously learned MAC Address. Thus, it is necessary to define the LT with the P4 language and

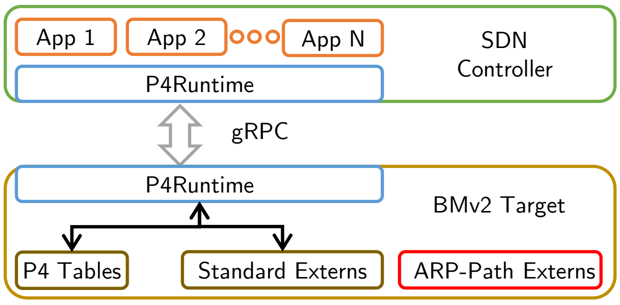


Figure 2 ARP-P4 non-standard extern methods limitation

later implement the ARP-Path logic to manage properly the LT. Taking into account these needs, one possible approach according to the P4 specifications would consist on embedding a P4-Runtime controller on each ARP-P4 device to supply these needs, but it does not seem to be a feasible solution for low-cost devices. Thus, ARP-P4 adopts an alternative solution, the use of non-standard P4 extern functions to insert, delete and update the LT. These extern functions are defined and implemented ad-hoc in each target. Thus, the selection of the BMv2 defines how these functions must be implemented to support the desired functionality. Therefore, an ARP-Patch match implies to check the extern LT, if an entry exists, the packet is forwarded to the learned egress port.

3.2 Hybrid Forwarding Engine

The construction of the ARP-P4 Hybrid Forwarding Engine is not easy because of the current limitations in the P4 language specification as well as the fact the required hybrid SDN ARP-Path/P4-Runtime behaviour. Figure 1 resumes the expected hybrid behaviour of ARP-P4. Therefore, it is necessary to support both the definition of the P4-Runtime and ARP-Path rules in an ARP-P4 device.

3.2.1 P4-Runtime Forwarding Support

P4-Runtime [9] allows the communication of the control plane, the controllers in an SDN architecture, with the data plane, the communication devices. Moreover, P4-Runtime allows the communication with data plane devices that are defined by the P4 language [8]. The SDN ONOS controller [10] supports P4-Runtime and posseses a development flow that allows the management of P4-based devices with P4-Runtime support due to the contributions of the P4 ONOS Brigade [16]. The whole integration process is documented, a good summary can be found in [17]. Indeed, it is only necessary to use a P4 compiler with P4-Runtime support, i.e. [18], to obtain the necessary P4INFO definition required by ONOS to make proper use of the P4 tables and to define the desired matching rules as well as their associated action rules. Hence, it remains how to define the ARP-Path Forwarding.

3.2.2 ARP-Path Forwarding Support

By definition, the P4 language does not include any way to modify P4-defined tables inside a P4 program. It must always rely on an external (or local independent) control plane to do this. This fact is a key aspect because ARP-P4 must not only define as externs the functions that implement ARP-Path, it must also define the LT as an extern data structure. The drawback of defining the LT as an extern data structure is the unawareness of the ARP-Path forwarding state by an external control plane, such as an SDN controller, due to the fact that the current specification of the P4 language is unable to interoperate with non-standard extern objects via P4-Runtime. This important undesired limitation is illustrated in Fig. 2 where ARP-Path externs cannot interact with P4-Runtime. Hence, ARP-P4 stores the learned MAC addresses in the LT, defined as a P4 extern data structure, in conjunction with its input port and a timestamp. This timestamp is set up with a locking short time if the entry is used to implement the ARP-Path locking mechanism that avoids loops. Moreover, the timestamp is configured with a learning longer time if the entry is confirmed to forward packets. Thus, ARP-P4 can forward packets with no-matching P4-Runtime rules by using the defined LT and the ARP-Path mechanism illustrated in Fig. 3. In order to perform this procedure properly, it is necessary an extern API with at least the following functions:

Flood manages broadcast traffic. ARP traffic is used to populate the LT with a locking short time. The rest of broadcast traffic can refresh the entries if the incoming port is the same one as the stored in the LT, which corresponds to the first ARP Request from the source address. This behaviour is depicted with polygons in red solid lines in Fig.3.

Forward performs the forwarding of unicast packets according to the entries in the LT. This behaviour is depicted with polygons in violet dot-dashed lines in Fig.3.

Reply forwards ARP Replies packets according to the entries in the LT and changes the timestamp to the learning longer time. This behaviour is depicted with polygons in blue dashed lines in Fig.3.

Num_entries: collects different statistics from the LT.

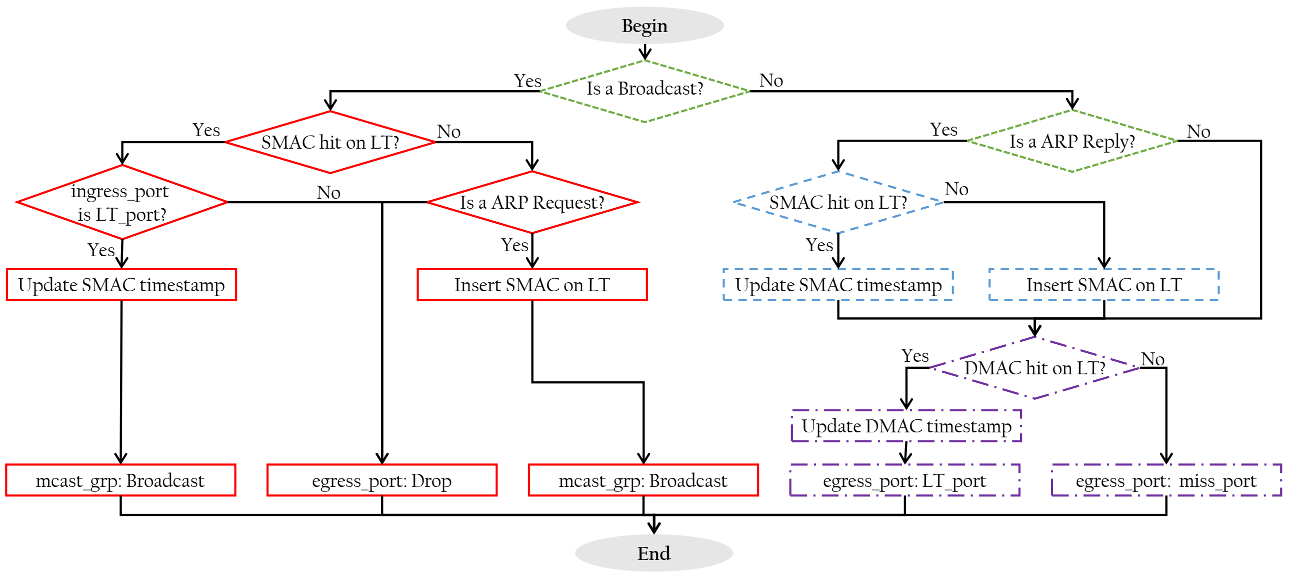


Figure 3 ARP-Path processing and forwarding of packets

Therefore, once the extern API is defined, this extern API can be used to define an autonomous P4 based pipeline that behaves according to the ARP-Path behaviour (see Fig. 3). Hence, the desired ARP-P4 Hybrid Forwarding Engine is finally functional since the mandatory blocks, P4-Runtime and ARP-Path from Fig. 1, are supported in spite of the encountered limitations.

3.3 BMv2 based target

Once the ARP-P4 behaviour is defined by using the P4 language, the following step is to introduce that behaviour in a target that allows to implement the P4 source code. The BMv2 software switch [19] has this functionality and also an API that supports the definition of extern data structures and private APIs. Moreover, this target also possesses a multicast group in the PRE to forward efficiently broadcast traffic, which is an important feature since ARP-Path establishes paths by exploring the shortest paths based on a controlled flooding mechanism. Unfortunately, the standard BMv2 target performs flooding sequentially among all configured ports except for the ingress port as expected (i.e. it always starts at the same port and go through the rest in the same order) by default. This behaviour is unfair since this predefined order can benefit the discovery of certain paths with respect to others. Thus, this sequential flooding has been replaced by a randomised one where each flooding action selects a random order of the egress ports to forward the packets. This randomisation mitigates the aforementioned effect. Hence, once the extern LT and its associated extern API are defined, an operational software ARP-P4 device is obtained.

4 Evaluation

In the evaluation of ARP-P4, we compare the obtained performance from both its ARP-Path autonomous behaviour and P4-Runtime behaviour with the performance of a legacy ARP-Path switch [7]. The P4-Runtime behaviour is configured by the use of an Equal-Cost Multi-Path (ECMP) routing scheme managed by an ONOS instance. Moreover, an additional test is conducted to demonstrate how both forwarding schemes can work together according to the proposed design.

4.1 Testbed

Our hardware infrastructure consists of 5 computers powered by Intel(R) Core(TM) i7 processors with 24 GB of RAM, all of which are interconnected via a GbE Netgear GS116 switch for emulation. To validate our ARP-P4 implementation, we use the *Mininet* [20] emulation platform, which allows the evaluation of ARP-P4 using a real Linux TCP/IP environment. The same emulation platform and network conditions are used with ARP-Path switches [7] to assure the same evaluation conditions.

4.2 Experimental Setup

ARP-Path establishes paths in Layer 2 domains since it does not make use of information from IP or upper layers, it uses exclusively Layer 2 information (MAC addresses). Therefore, high demanding Layer 2 scenarios

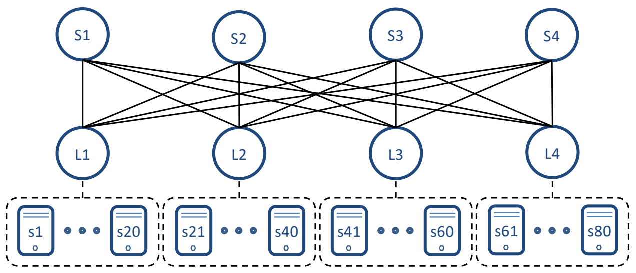


Figure 4 Spine-Leaf 4-4-20 evaluation topology

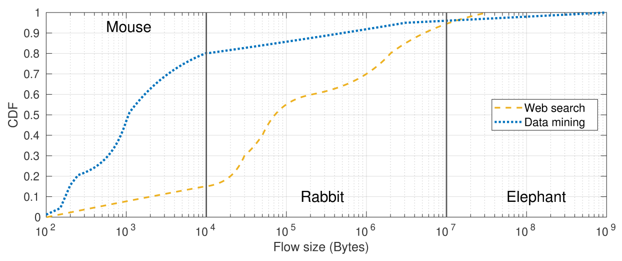


Figure 5 Flow size CDFs

are excellent candidates two evaluate ARP-P4 or ARP-Path such as data center scenario. *Spine-Leaf* topologies [21–23] are typically deployed for high-performance scenarios such as the aforementioned data center networks. Thus, this kind of topology is selected to perform the experiments since it is representative and widely used. Hence, a 4-4-20 Spine-Leaf topology, which contains 2 rows of 4 switches (4 *spines* and 4 *leaves*) with 20 servers per leaf switch for a total of 80 servers, is used for the evaluation of ARP-P4. Figure 4 shows a scheme of the topology setup.

Traffic flows are randomly distributed between any pair of servers attached to two different leaf switches with no further restrictions. In addition, we consider two different flow size distributions, *Data Mining* and *Web Search*, derived from experimental traces taken from actual data center networks [24, 25]. Fig. 5 shows the CDF of both distributions and also illustrates how flows are classified according to their size in elephants, rabbits and mice. Flows with less than 10 KB and more than 10 MB of data are considered *mouse* and *elephant* flows, respectively, as explained in [24]. The remaining flows are identified as *rabbit* flows. Figure 6 depicts the percentage of flows of each type (left), as well as the percentage of transmitted bytes by each of them (right). Lastly, we calculate the average *flow inter-arrival time* (IAT) to achieve an average offered network load of 10%, 20%, and 40% with respect to the full capacity of links, according to either the Web search or Data mining flow size distributions.

Each experiment runs for 1800 seconds and it is repeated 10 times to later compute 95% confidence inter-

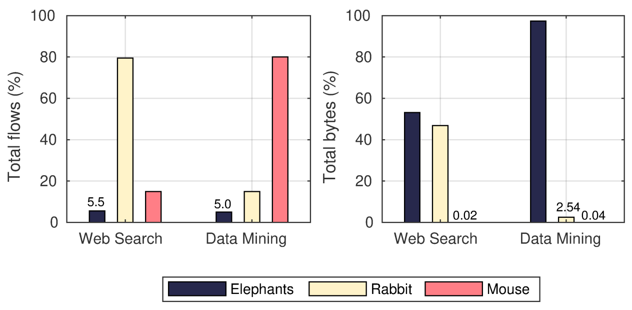


Figure 6 Total flows and total bytes per flow type

Table 1 Experimental Setup

Parameter	Value
Network topology	Spine-Leaf (4 - 4)[21]
Servers per leaf switch	20
Flow distribution	Random inter-leaf
Flow size CDFs	Web search[25], Data mining [24]
Network offered load	10, 20 & 40%
Link speed (Mpbs)	10Mbps
Run length (s)	1800 s
Warm up time (s)	800 s
Number of runs	10
Confidence intervals	95%

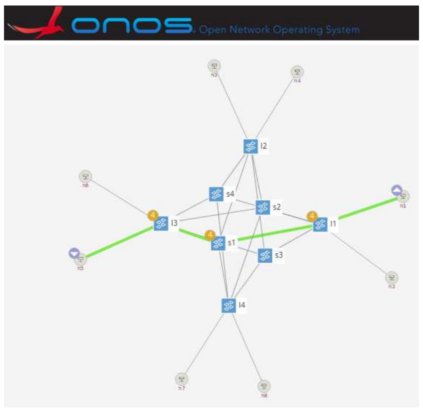
vals. Additionally, we consider a warm-up time of 800 seconds to mitigate any transitory effect on the results. Table 1 summarises the full setup of the conducted experiments.

4.3 Results

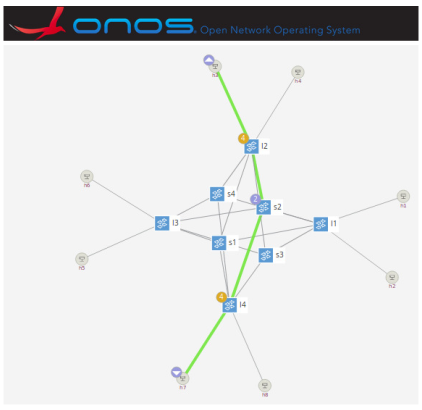
We monitored the the ARP-P4 devices by using the ONOS platform since it has P4-Runtime support. Hence, it was necessary for this task the required pipeconf configuration from the ARP-P4 P4INFO definition, see section 3.2.2, to interoperate with the ARP-P4 devices. Two different sets of results have been obtained. The first set of results measures the performance of ARP-P4 devices when they establish paths using ARP-Path or P4-Runtime, where an ECMP policy is used. The second set of results verifies the hybrid SDN ARP-Path/P4-Runtime forwarding.

4.3.1 ARP-P4 Performance

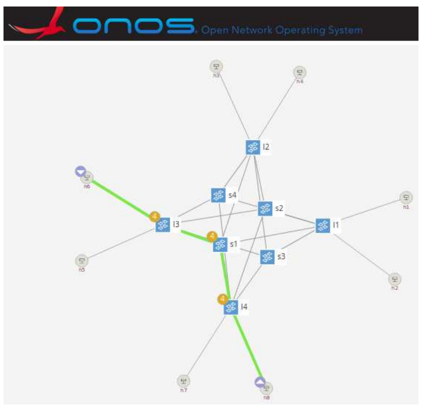
On the one hand, Fig. 7 shows different monitored paths in green, which are obtained autonomously from our ARP-P4 devices, which demonstrates their ARP-Path functionality. On the other hand, Fig. 8 shows the performance achieved by ARP-P4. It is measured the performance when ARP-P4 establishes paths using ARP-Path, ARP-P4 (ARP-Path) results, or P4-Runtime rules,



(a) First path

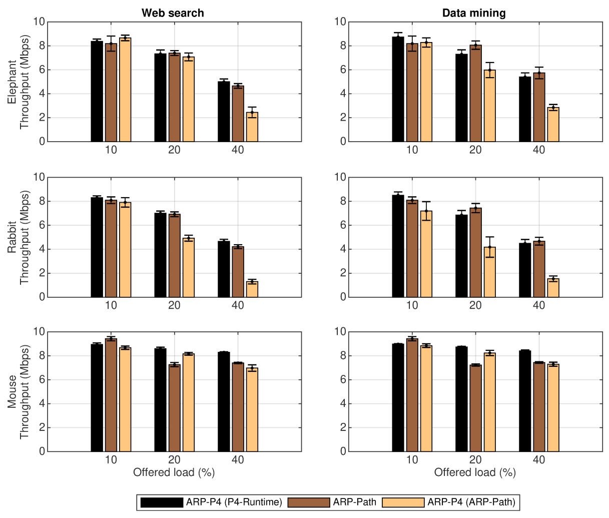


(b) Second path

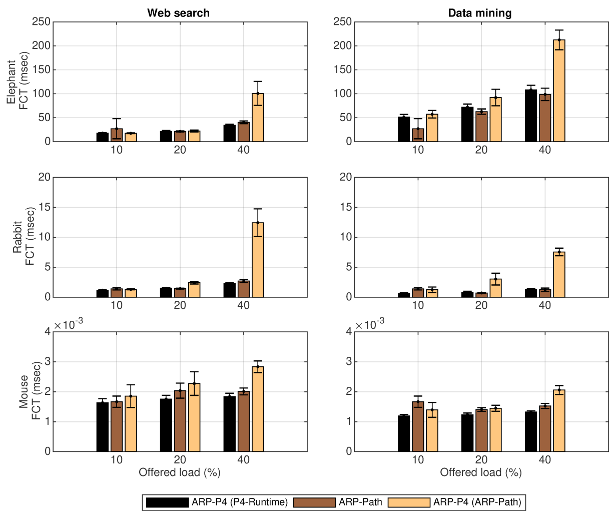


(c) Third path

Figure 7 Paths Diversity on Spine-Leaf (4-4-20) topology



(a) Throughput on Spine-Leaf topology



(b) Flow Completion Time on Spine-Leaf topology

Figure 8 Performance measurements on Spine-Leaf (4-4-20) topology

ARP-P4 (P4-Runtime) results. If P4-Runtime is used, paths are set up by an ONOS controller by applying an ECMP policy to a k-shortest path algorithm) for the different types of flows (*elephant*, *rabbit*, and *mouse*). Moreover, the results are also compared with other software switch that also supports ARP-Path [7], ARP-Path results.

Figure 8(a) depicts the obtained throughput of ARP-P4 devices. The results obtained using the Web Search traffic distribution are shown on the left-hand side of the figure and those obtained using the Data Mining distribution on the right-hand side. Throughput decreases with the offered load regardless of the protocol and traffic distribution in use as expected. When the number of flows increases, each flow can get a smaller portion of the available resources (link capacity). This produces that the flows obtain a smaller bandwidth, therefore their flow complexion time (FCT) increases (see Fig.8(b)) as well as their time in the network, which causes a decrease in the performance of the network. ARP-Path without a global controller and using only the local information has a very similar performance than using ECMP with P4-Runtime. This behaviour can be observed in Fig. 8. The result is quite relevant since a performance similar to the use of ECMP with P4-Runtime is possible without the intervention of a controller in a distributed and simple way. Moreover, it is important to remember how is possible to establish rules from an SDN controller with higher priority, to achieve any other required policies if necessary, due to its hybrid properties.

When the network load is low, we observe how the performance of ARP-P4 when uses ARP-Path is very similar to the other ARP-Path switch but, when the load increases, the performance of ARP-P4 decreases. On the one hand, this fact can be observed in Fig. 8(a) where the throughput decreases unexpectedly when the offered load is equal to 40%. On the other hand, the complementary effect is observed in Fig. 8(b), where the FCT suddenly increases if the load reaches 40%. This fall in performance on both measurements is more accentuated with elephant and rabbit flows (regardless of the flow size distribution), which are the main contributors to the offered load in the conducted experiments. Thus, we revised again if the setup of paths by ARP-P4 were correct. No issue was found; thus, we concluded that some kind of bottleneck or issue might exist in the ARP-P4 enabled target that degrades the expected performance. To explain this unexpected low performance, we analysed the code of BMv2 in deep to look for some explanation. Finally, we found out that the packets are not processed in the same order as their arrival order by a BMv2 target. A BMv2 target dequeues one packet from each ingress port following a Round Robin (RR) policy. This RR policy provokes a disorder in the received packets and masks the real delays of the explored paths, making the ARP-Path exploration process in the target not to work properly since the fastest paths cannot be found due to this issue. This undesirable effect is especially noticeable on high loads with lots of packets in the ingress queues.

4.3.2 Hybrid SDN ARP-Path/P4-Runtime functionality test

The remaining test related to ARP-P4 is to validate that both ARP-Path and P4-Runtime forwarding engines work together properly. In order to check this behaviour, a Packet Test Framework (PTF) [26] is used, which is based on Python and the Scapy library [27]. This PTF allows the definition of unity tests in devices with P4-Runtime support. A set of rules is set up via P4-Runtime through the PTF. Moreover, a set of ingress packets is injected in the ARP-P4 device and it is checked if we obtain the expected output from the ARP-P4 device. Specifically, an ACL is defined by the unity test and consequently, through P4-Runtime, no forwarding rules are defined, they are only filtering rules. Hence, if the P4-Runtime rules work properly, packets would be dropped if they do not match the ACL. In addition, no egress packets should exist if the ARP-Path autonomous forwarding does not work since the configured P4-Runtime rules cannot forward packets. Therefore, the unity test much check that the

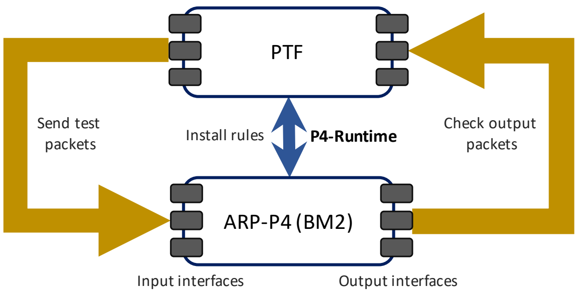


Figure 9 Unity Test

undesired traffic is blocked and how the desired traffic is forwarded through the egress port as expected according ARP-Path (P4-Runtime rules are not setup to forward packets). Hence, we use the PTF as a P4-Runtime client to verify that the ACL works properly. Figure 9 shows how the PTF first installs the ACL filtering rules to verify later the ARP-P4 hybrid behaviour by injecting well-known packets and checking if the packets are forwarded or dropped as expected.

The unity test is defined as follows. An ARP-P4 devices is used to connect an internal network with IP prefix 10.0.0.0/16, a Demilitarized Zone (DMZ) network with IP prefix 160.88.82.0/24 that supports two (DNS and HTTP) servers, a external network with IP prefix 160.20.0.0/16 and a gateway that gives access to Internet. Furthermore, a SDN controller monitors and restricts dynamically the employees traffic by using the ARP-P4 device. Only certain IP prefixes and domains are allowed, which form a white-list. Therefore, the polices according the described scenario are desired as follows: (1) All the traffic inside the IP network 10.0.0.0/16 is allowed. (2) TCP/UDP traffic from inside to outside is permitted. (2) DNS and HTTP traffic towards the DMZ network is allowed. (4) The traffic between the aforementioned IP prefixes is also permitted. (5) TCP SYN packets from inside are forwarded to the SDN controller to allow or discard the flows according the white-list. (6) Any other traffic is blocked.

After installing the policy rules via P4-Runtime, the unitary test injects the following packets. (1) A TCP SYN and a UDP datagram from outside to the internal network. Both packets are discarded by the P4-Runtime rules. (2) A DNS query and a HTTP access from the outside to the DMZ network. Both requests are allowed by the P4-Runtime rules and later forwarded by using ARP-Path. (3) TCP SYN from inside generates a packet_in in the SDN controller. If the destination address does not match the white-list, the packet is discarded in the controller, otherwise the packet is sent back via a packet_out since it is allowed. Later, the packet is forwarded by using ARP-Path according the behaviour defined in Fig. 1. (4) TCP SYN packets from internal networks target the considered external network. The opposite direction is also considered. All requests are allowed by the P4-Runtime rules and later forwarded by using ARP-Path again. (5) ICMP packets from outside are sent inside. All of them are discarded by the P4-Runtime rules. Hence, whole unity test behaves as expected according the desired policies. Thus, the expected hybrid ARP-Path/P4-Runtime behaviour of ARP-P4 is confirmed since the authorised packetes are properly forwarded.

5 Current ARP-P4 limitations

On the one hand, after studying thoroughly the behaviour of ARP-P4 as well as its definition in BMv2 target, we can see the limitations of its design. Although ARP-P4 behaves as expected, it is undesirable that the designed P4 based device with autonomous control plane capabilities cannot access, manage and modify standard P4 matching tables according to the inspection of ingress packets. Currently, it is only possible to operate with P4 matching tables through P4-Runtime. This is an important constraint since no simple policy can be established to forward packets autonomously, which make no possible the definition of a simple learning or an ARP-Path switch. We believe that the P4 language should be extended not only with the objective to fulfil the requirements of a date plane for an SDN architecture based on a remote control plane, but P4 should also be able to allow also the description of a local control plane if desired. Therefore, the definition of primitives on P4 that allow the management of standard P4 matching tables via P4-Runtime from internal P4 actions or extern objects would be the key to provide to the P4 language of more functionalities and higher flexibility. Furthermore, it is important to highlight that autonomous behaviours do not imply isolation with respect to SDN control planes, i.e. ARP-P4 is envisioned to support both control planes.

On the other hand, the performance issues that the BMv2 based targets currently suffer are motivated by how ingress packets are processed because their order of arrival is not maintained for their later processing. This fact has an important impact on ARP-P4 performance since the queuing time of packets is modified and the ARP-Path protocol is based on the packet delays suffered by packets to discover the path with less congestion. Moreover, this issue could also affect to any Quality of Service (QoS) policy established either autonomously or via a P4-Runtime capable SDN controller. Perhaps other targets will not suffer this issue.

6 Conclusions

The ARP-P4 switch is a hybrid SDN ARP-Path/P4-Runtime switch that implements both ARP-Path and P4-Runtime based behaviours by using the P4 language. ARP-P4 devices can autonomously forward packets in a level 2 domain if no P4-Runtime matching entries exist. Currently, it has only be implemented in a BMv2 target using a P4 program that leverage non-standard extern functions, without a local controller, to allow lightweight and low-cost devices. The hybrid SDN ARP-path/P4-Runtime behaviour has been successfully verified with an unity test using a PTF. Hence, SDN controllers can reduce their load if they delegate basic forwarding tasks to ARP-P4 devices. Moreover, the performance of ARP-P4 has been demonstrated effective on both ARP-Path and ECMP P4-Runtime based forwarding when a controller establishes the ECMP paths. Unfortunately, the ARP-Path performance of ARP-P4 is below the expectations, mainly due to the fact that BMv2-based targets do not maintain the arrival time of the packets, which causes ARP-P4 devices to misbehave, particularly if they forward packets using ARP-Path. Therefore, the problem does not come from the ARP-P4 design, but from the BMv2 implementation. To conclude, we can say that ARP-P4 is a successful attempt to push P4 towards its own limitations by defining a data plane with autonomous behaviour based on ARP-Path; although, P4-Runtime and the ARP-Path extern objects and functions cannot yet directly interact due to the current limitations of the P4 language.

7 Future Work

Different research lines can arise from the analysis of the design and evaluation of ARP-P4. The current limitations of the P4 language are trully a challenge to overcome. On the one hand, it would be interesting to make available non-standard externs to P4-Runtime. On the other hand, it would be also desirable to allow extern functions to modify P4 tables and sync any modifications with P4-Runtime to allow a fully transparent and synced state among P4 devices with their associated SDN controllers, particularly if they present autonomous behaviour like ARP-P4. Finally, it would be also potentially useful to define some unity tests to evaluate the behaviour of different targets supporting P4 as well as performance tests to identify bottlenecks or undesired inefficiencies in a P4 based pipeline, so the target development and evolution would be strengthened.

References

1. 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*, 103(1):14–76, Jan 2015. ISSN 0018-9219. doi: 10.1109 / JPROC.2014.2371999.
2. Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. OpenFlow: Enabling Innovation in Campus Networks. *SIGCOMM Comput. Commun. Rev.*, 38(2):69–74, March 2008. ISSN 0146-4833. doi: 10.1145 / 1355734.1355746.
3. E. Rojas. From Software-Defined to Human-Defined Networking: Challenges and Opportunities. *IEEE Network*, 32(1):179–185, Jan 2018. ISSN 0890-8044. doi: 10.1109/MNET.2017.1700070.
4. Isaias Martinez-Yelmo, Joaquin Alvarez-Horcajo, Miguel Briso-Montiano, Diego Lopez-Pajares, and Elisa Rojas. ARP-P4: A Hybrid ARP-Path/P4Runtime Switch. In *1st P4 Workshop in Europe (P4WE) in proceedings of 2018 IEEE 26th International Conference on Network Protocols (ICNP)*, pages 438–439. IEEE, 2018.
5. R. Amin, M. Reisslein, and N. Shah. Hybrid SDN Networks: A Survey of Existing Approaches. *IEEE Communications Surveys Tutorials*, pages 1–1, 2018. doi: 10.1109/COMST.2018.2837161.
6. Giuseppe Bianchi, Marco Bonola, Antonio Capone, and Carmelo Cascone. OpenState: Programming Platform-independent Stateful Openflow Applications Inside the Switch. *SIGCOMM Comput. Commun. Rev.*, 44(2):44–51, April 2014. ISSN 0146-4833. doi: 10.1145/2602204.2602211.
7. Joaquin Alvarez-Horcajo, Isaias Martinez-Yelmo, Elisa Rojas, Juan A. Carral, and Diego Lopez-Pajares. New cooperative mechanisms for software defined networks based on hybrid switches. *Transactions on Emerging Telecommunications Technologies*, 28(8):e3150, 2017. doi: 10.1002 / ett.3150. e3150 ett.3150.
8. Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, and et al. Vahdat. P4: Programming Protocol-independent Packet Processors. *SIGCOMM Comput. Commun. Rev.*, 44 (3):87–95, July 2014. ISSN 0146-4833. doi: 10.1145/2656877.2656890.
9. Nick McKeown, Timon Sloane, and Jim Wanderer. P4 runtime - putting the control plane in charge of the forwarding plane, 2017. Available at https://p4.org/api/p4-runtime-putting-the-control-plane-in-charge-of-the-forwarding-plane.html (Access date: 2018-06-22).
10. Pankaj Berde, Matteo Gerola, Jonathan Hart, Yuta Higuchi, Masayoshi Kobayashi, Toshio Koide, Bob Lantz, Brian O'Connor, Pavlin Radoslavov, and et al. Snow. ONOS: Towards an Open,Distributed SDN OS. In *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, 2014.
11. P4 Consortium. BMv2: Designing your own switch target with BMv2, 2018. Available at http://www.bmv2.org (Access date: 2018-06-13).
12. G. Ibanez, J. A. Carral, J. M. Arco, D. Rivera, and A. Montalvo. ARP-Path: ARP-Based,Shortest Path Bridges. *IEEE Communications Letters*, 15 (7):770–772, July 2011. ISSN 1089-7798. doi: 10.1109/LCOMM.2011.060111.102264.
13. David Allan and Nigel Bragg. *802.1 aq Shortest Path Bridging Design and Evolution: The Architects' Perspective.* Wiley Online Library, 2012.
14. R. Perlman, D. Eastlake 3rd, D. Dutt, S. Gai, and A. Ghanwani. Routing Bridges (RBridges): Base Protocol Specification, July 2011. RFC6325. Available at http://tools.ietf.org/rfc/rfc6325.txt (Access date: 2018-09-20).
15. P4 Consortium. Behavioral Model: Rewrite of the behavioral model as a C++ project), June 2018. Available at https://github.com/p4lang/behavioral-model (Access date: 2018-06-04).
16. P4 brigade - ONOS - Wiki, 2019. Available at https://wiki.onosproject.org/display/ONOS/P4+brigade#P4brigade-Learnmore (Access date: 2019-04-29).
17. ONOS+P4 Tutorial - ONOS - Wiki, 2019. Available at https://wiki.onosproject.org/pages/viewpage.action ?pageId=16122675 (Access date: 2019-04-29).
18. P4_16 prototype compiler. Contribute to p4lang/p4c development by creating an account on GitHub, April 2019. Available at https://github.com/p4lang/p4c (Access date: 2019-04-29).
19. P4 Consortium. Simple Switch Grpc - A version of SimpleSwitch with P4 Runtime support, June 2018. Available at https://github.com/p4lang/behavioral-model/tree/master/targets/simple_switch_grpc (Access date: 2018-06-15).
20. Nikhil Handigol, Brandon Heller, Vimalkumar Jeyakumar, Bob Lantz, and Nick McKeown. Reproducible network experiments using container-based emulation. In *Proceedings of the 8th Interna-*

tional Conference on Emerging Networking Experiments and Technologies, CoNEXT '12, pages 253–264, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1775-7. doi: 10.1145/2413176.2413206. URL `http://doi.acm.org/10.1145/2413176.2413206`.

21. Mohammad Alizadeh, Tom Edsall, Sarang Dharmapurikar, Ramanan Vaidyanathan, Kevin Chu, Andy Fingerhut, Vinh The Lam, Francis Matus, Rong Pan, and et al. Yadav. CONGA: Distributed Congestion-aware Load Balancing for Datacenters. *SIGCOMM Comput. Commun. Rev.*, 44(4):503–514, August 2014. ISSN 0146-4833.
22. Keqiang He, Eric Rozner, Kanak Agarwal, Wes Felter, John Carter, and Aditya Akella. Presto: Edge-based load balancing for fast datacenter networks. *SIGCOMM Comput. Commun. Rev.*, 45 (4):465–478, August 2015. ISSN 0146-4833. doi: 10.1145/2829988.2787507.
23. Mohammad Alizadeh, Shuang Yang, Milad Sharif, Sachin Katti, Nick McKeown, Balaji Prabhakar, and Scott Shenker. pFabric: Minimal Near-optimal Datacenter Transport. *SIGCOMM Comput. Commun. Rev.*, 43(4):435–446, August 2013. ISSN 0146-4833. doi: 10.1145/2534169.2486031.
24. Albert Greenberg, James R. Hamilton, Navendu Jain, Srikanth Kandula, Changhoon Kim, Parantap Lahiri, David A. Maltz, Parveen Patel, and Sudipta Sengupta. VL2: a scalable and flexible data center network. *SIGCOMM Comput. Commun. Rev.*, 39 (4):51–62, 2009.
25. Mohammad Alizadeh, Albert Greenberg, David A. Maltz, Jitendra Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan. Data center TCP (DCTCP). *SIGCOMM Comput. Commun. Rev.*, 41(4):63–74, August 2010. ISSN 0146-4833.
26. GitHub - p4lang/ptf: Packet Test Framework, 2019. Available at https://github.com/p4lang/ptf (Access date: 2019-04-29).
27. Scapy, 2019. Available at https://scapy.net/ (Access date: 2019-04-29).