Álvarez Horcajo, J., Martínez Yelmo, I., López Pajares, D., Carral, J.A. & Savi, M. 2021, "A hybrid SDN switch based on standard P4 code", IEEE Communications Letters, vol. 25, no. 5, pp. 1482-1485

Available at http://dx.doi.org/10.1109/LCOMM.2021.3049570

A Hybrid SDN Switch based on Standard P4 Code

Joaquin Alvarez-Horcajo, Isaias Martinez-Yelmo, Diego Lopez-Pajares, Juan A. Carral, and Marco Savi

***Abstract*—This paper presents an enhanced hybrid Software-Defined Networking (SDN) layer-2 switch whose behavior is specified by the Programming Protocol-independent Packet Processors (P4) language. Its SDN capabilities are enabled by using P4Runtime as control plane protocol to specify the forwarding rules used by its programmable data plane. Additionally, the device is also able to exploit P4 registers for an autonomous self-definition of its forwarding capabilities, with the goal of avoiding an overload of the SDN control plane. Its performance is better than other P4 proposals based on non-standard externs and similar to other platform-dependent implementations.**

***Index Terms*—P4, ARP-Path Protocol, P4 Registers, Forwarding Tables, Autonomous Path Selection**

I. Introduction

SINCE the SDN architecture emerged [1], network programmability is in continuous evolution. The SDN paradigm started by defining how to program an independent control plane, used to communicate with the data plane by using the OpenFlow protocol. However, nowadays the focus has broaden to also make the data plane programmable. This fact has led to the definition of the P4 language [2], which enables a highly-programmable processing and manipulation of data plane packet headers by following some rules as installed by the SDN control plane through *P4-Runtime*, which is the natural replacement of OpenFlow in P4-enabled devices.

To ensure interoperability, these advancements have been often coupled with the design of *hybrid* SDN solutions [3]. A hybrid SDN switch is a device that is capable of both working as demanded by the SDN control plane and of taking autonomous decisions without requiring any interaction with it (like legacy devices). This is very useful if the control plane is only interested in managing *premium* services (due to priority level agreements or security reasons), but does not care/have resources to manage them all. Work in [4] shows that hybrid SDN switches, capable of autonomously applying self-learning forwarding rules if control plane rules do not match, alleviate the load at the control plane (since exchanged packets and processing needs are reduced).

Recently, a hybrid SDN switch solution leveraging the P4 language has been proposed [5]. Unfortunately, such work uses non-standard extensions of P4 that cripple the portability of the code and prevent its wide adoption and deployment when heterogeneous P4 targets have to execute the P4 program. To overcome this limitation, this article focuses on the use of *P4 registers*, which are standard data structures present in P4 specifications, to define and implement hybrid SDN switches. Such data structures are used to store state information, with the aforementioned goal of enabling autonomous forwarding in the data plane without the intervention of a control plane. Unlike previous works [5], our proposal uses standard P4-supported data structures, having the advantage of being fully portable and executable by any P4 target, and achieving the same level of offloading of the control plane tasks as other hybrid SDN devices [4]. Moreover, it makes the deployment in networks with heterogeneous P4 devices easier.

The paper is organised as follows. First, we examine the related work in Section II. Secondly, we describe our design and implementation and its evaluation in Sections III and IV. Finally, the conclusions of the work are in Section V.

II. Related work

The SDN [1] paradigm proposes moving the network intelligence from the network devices to a logically centralised control plane that controls and orchestrates the underlying network. However, the SDN architecture requires a high control message overhead [6] and scalability issues could arise [7], [8]. Possible solutions range from running in parallel multiple synchronised controller instances as in [9], to implement controller offloading techniques such as delegating or reverting back certain control plane tasks to the data plane [3]. The latter can be accomplished by using *hybrid* SDN devices with autonomous forwarding capabilities that can be enabled or disabled by the control plane [4] and permits new cooperative mechanisms between the control and data plane if all hybrid SDN devices possess the same autonomous capabilities. This is a cumbersome requirement and limitation unless it comes coupled with data plane programmability features. Indeed, there is an important effort around the P4 language [2], so that it can be adopted as a standard language to program data plane functionalities. Although it was originally designed to support full-SDN architectures, it can also be used to define hybrid P4 SDN devices, as done in [5]. However, such work requires the usage of non-standard P4 externs, which are platform-dependent data structures and functions that ask for ad-hoc implementations depending on the P4 target in use (e.g. Behavioral-Model version 2 (BMv2) devices [10] or SUME NetFGPAs [11]). The use of non-standard P4 externs is not uncommon [12] and comes from the impossibility of modifying the P4 forwarding tables while processing packets in the P4 pipeline, replacing the need of relying on the SDN controller for the injection of new rules. Unfortunately, being non-standard extensions of the language, a widespread adoption of extern-based solutions is not achievable.

Joaquin Alvarez-Horcajo, Isaias Martinez-Yelmo, Diego Lopez-Pajares and Juan A. Carral are with Departamento de Automatica, University of Alcala, Alcala de Henares, Spain. Marco Savi is with University of Milano-Bicocca, Milano, Italy. The study was partially done while he was with Fondazione Bruno Kessler, Trento, Italy. Corresponding author's e-mail: isaias.martinezy@uah.es

This work was funded by grants from Comunidad de Madrid: projects TAPIR-CM (S2018/TCS-4496) and IRIS-CM (CM/JIN/2019-039), from Junta de Comunidades de Castilla la Mancha: project IRIS-JCCM (SBPLY/19/180501/000324), and from University of Alcalá: "Programa de Formación del Profesorado Universitario-FPU" and project CCG2018_EXP-076.

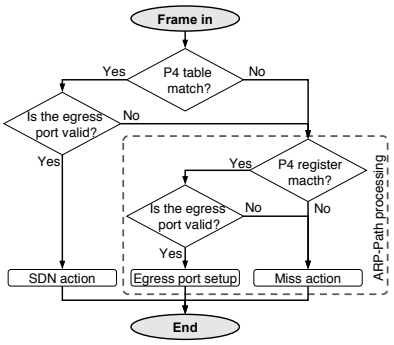


Fig. 1. Layer-2 forwarding (high-level design)

This paper shows how hybrid SDN devices implemented in fully-portable P4 code can be designed, by using P4 standard registers as temporary-indexed data structures [13], [14] to store the forwarding information obtained by the P4 data plane while processing packets, without the intervention of the control plane. Hence, the use of P4 standard registers is the key for enabling P4 code portability, since it makes it possible to overcome the main limitation of non-standard P4 externs.

III. Portable P4 design of a hybrid SDN switch

In this section we present our proposed solution. The P4 registers are used to store the data structures needed for an autonomous forwarding of data plane packets (i.e., an *autonomous forwarding table*); these structures would be later populated by the distributed forwarding protocol. Without any loss of generality, we choose to implement ARP-Path Protocol (ARP-Path) [15], [5], as autonomous forwarding protocol, for comparison purposes. Moreover, the centralised SDN capabilities remain intact since the control plane can still operate the devices by using *P4-Runtime* [16] to configure SDN rules on regular P4 match-action tables. These rules take precedence over data-plane ones by architectural design.

A. Architectural design

As mentioned above, the proposed architectural design gives higher priority to control plane rules. This is shown in Fig. 1, which describes how incoming packets are handled. Upon a packet arrival, first, the device looks for any valid *P4-Runtime* rule. If a rule exists, which means that a P4 match-action table *hit* occurs, and the egress port is valid, the forwarding strategy applies the action associated with the rule to the ingress packet. Only if no valid egress port is available, the strategy looks at a *P4 register array* to find an egress port to forward the packet. In the case that a rule exists and the egress port is valid, the device forwards the packet to the egress port indicated in the matched P4 register. If neither the P4 match-action table nor the P4 register array contain a valid rule to apply, a *miss* action is required: unless an optional recovery mechanism exists, the miss action should drop the packet.

The key element for packet handling is how to define and manage the data structures to support the matching entries and their actions in P4 registers. But, before that, it is necessary to define how to populate them with data-plane derived forwarding information by using ARP-Path, so a minimal background on ARP-Path protocol is provided in the following section.

B. Background on ARP-Path protocol

ARP-Path [15] is a layer-2 protocol for Ethernet-bridged networks. It relies on the Address Resolution Protocol (ARP) packet exchange to discover and set up low-latency paths. ARP-Path works in two phases: the *exploration phase*, which relies on the *ARP Request* packet broadcast to discover low-latency paths to the source node (in fact the paths discovered form a sink tree rooted at the source node and spanning to every other node in the network), and the *confirmation phase*, which relies on the *ARP Reply* packet to set up the path between the pair of source and destination nodes (i.e., it confirms the tree branch from source to destination node, which was previously discovered during the exploration phase). The protocol works as follows: every node receiving an *ARP Request* packet associates the corresponding source Media Access Control (MAC) address with the input port where it was received and stores this information in a timed Blocking Table (BT). Other copies (late copies) of the same *Request* are simply discarded (based on the information on the BT) to prevent loops. Eventually, at least one copy of the *Request* packet reaches every other node in the network, including the forwarding node serving the requested end point, which replies with the corresponding *ARP Reply* packet. Now, every node receiving a *Reply* packet copies the corresponding entry of the BT to another table, called Learning Table (LT), by also including in LT a second entry that associates the source MAC address of the *Reply* packet with its corresponding arrival port, thus setting the path to both source and destination MAC addresses in the considered node. Afterwards, it simply forwards the *Reply* packet to the destination MAC address. Moreover, LT entries are refreshed whenever a unicast frame is forwarded, to prevent their aging. To simplify the protocol implementation, both BT and LT tables can be merged into a single table using shorter timeouts for blocking entries and longer timeouts for learning entries.

C. Design of ARP-Path using P4 registers

This section presents the ARP-Path-based P4 implementation of the autonomous forwarding capabilities of the proposed layer-2 switch. The switch's data plane implements the ARP-Path forwarding logic by using two different arrays of P4 registers, the *port register array* and the *time register array*, to save the state of the protocol. Together, they play the role of the BT and LT tables mentioned above. The port register array saves the ingress port from ingress frames by taking the source MAC address as register index, to later forward ingress packets according to their destination MAC address. The time register array saves timestamps instead, indicating the expiration times of the port register array entries. Both arrays jointly work as a unique table since they use the same index (see Fig. 2) for register population.

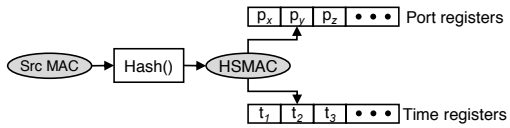


Fig. 2. Time and port P4 register arrays

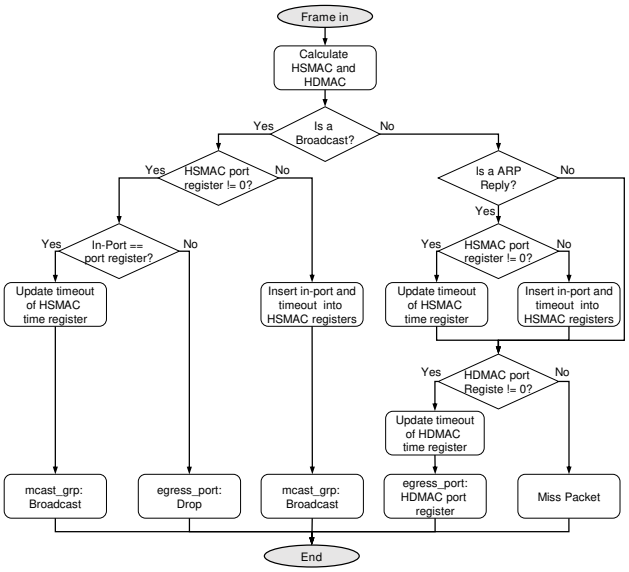


Fig. 3. ARP-Path frame processing using P4 registers

Figure 3 shows the flowchart that summarises the proposed design. The process starts when a frame arrives at a hybrid SDN device with no match on an egress port after applying the existing P4 matching rules. First, the node applies an *identity hash function* on the source and destination MAC addresses to calculate the Hash Source Media Access Control (HSMAC) and the Hash Destination Media Access Control (HDMAC) values. Such hash function is based on the modulo operation, which makes it possible to set the output size of the hash function equal to the size of the port and time registers, so that all the register's cells can be indexed. The HSMAC and HDMAC values are used as search indexes within the P4 register arrays (see Fig. 2). Then, the switch applies the ARP-Path protocol logic based on the type of received frames.

The left-hand side of the chart shows the processing if a broadcast or multicast frame is received. The strategy looks for the corresponding HSMAC on the port register array: if a valid entry is found and it points to the frame incoming port, the frame is simply broadcast/multicast and the corresponding timestamp, in the time register array, is updated. Otherwise, if the port register array entry points to a different port than the incoming port, the frame is discarded to prevent loops. When no valid entry for HSMAC is found on the port register, the switch associates the HSMAC with the incoming port and stores this information in both register arrays, then it simply broadcasts/multicasts the frame.

The right-hand side of the chart shows the processing of a unicast frame. First, the HSMAC is processed. In the case that the frame is an ARP *Reply* and there is no valid entry for HSMAC in the port register, the switch creates a new entry associating the HSMAC to the corresponding arrival port in both registers. If a valid entry already exists, it simply updates the corresponding time register. Then, the HDMAC is processed in a similar way as the HSMAC. If a valid entry exists in the port register, the switch forwards the frame through the corresponding egress port and updates the timeout. Otherwise, the frame is discarded.

IV. Evaluation

Following the same scheme as in [5], we evaluated ARP-Path-based forwarding proposal based on *P4 registers* (ARP-P4-Reg) and shown in Section III-C, in terms of throughput and Flow Completion Time (FCT), which are the traditional metrics in data center networks requiring efficient layer-2 forwarding capabilities. We provide a performance comparison of ARP-P4-Reg, the implementation proposed in [5] and based on P4 *externs* (ARP-P4-Ext), an ad-hoc non-P4 hybrid SDN device [4] (ARP-AdHoc) and a traditional SDN device running a Equal-Cost Multi-Path (ECMP) routing policy. Both P4-based solutions, i.e., ARP-P4-Reg and ARP-P4-Ext, are evaluated using BMv2 software devices [10] with P4 support.

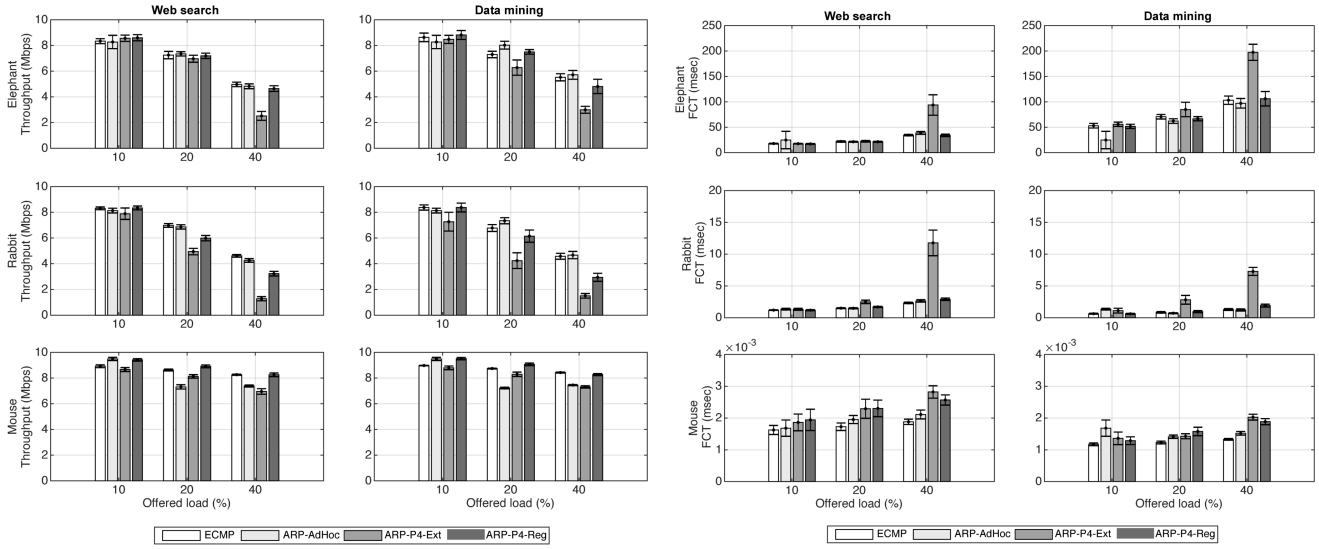
A. Experimental Setup

The testbed is composed of 4 Intel(R) Core(TM) i7 servers running Mininet [17] as emulation platform. To carry out our evaluation, we adopt the same scenario as in [5]: a *Spine-Leaf* topology [18] made of 4 *spines* and 4 *leaves* with 20 servers per leaf switch, for a total of 80 servers. The traffic flows are generated from a random traffic matrix where the source and destination of each flow must be on different *leaf* nodes. Flow sizes are defined from two cumulative distributions functions (CDFs), i.e., *Data Mining* [19] and *Web Search* [20], both obtained from real data center traces. Finally, we set the flow Inter Arrival Time (IAT) to reach an average offered network load of 10%, 20%, and 40% with respect to the full capacity of links (10Mbps due to testbed constraints). Each experiment runs for 1800s (a warmup time of 800s is considered) and is repeated 10 times to compute 95% confidence intervals.

For ARP-P4-Reg, we set 327680 cells as size of each port and time register array. Such size, in line with the maximum possible as per specification of existing carrier-grade programmable switches, is big enough to ensure a low number of hash collisions in the considered scenario, thus causing a negligible probability of forwarding errors.

B. Results

Figure 4(a) shows the result comparison between the considered strategies for different types of flows in terms of throughput, while Fig. 4(b) shows the same comparison in terms of FCT. The results shown on the left-hand side of each figure correspond to the Web Search, while the ones on the right-hand side to the Data Mining distribution. We can see that the new design, based on P4 registers, clearly outperforms ARP-P4-Ext. The use of standard mechanisms, more tested and optimised, are behind the experienced improvements. Moreover, the new proposal has smaller complexity than ARP-P4-Ext due to the use of registers as autonomous forwarding tables; also, the P4 *extern* handler of BMv2 introduces a higher delay in packet handling than the standard P4 handler. Related to the other implementations, the performance of the new design is similar to ECMP or traditional ARP-Path switches for *elephant* (big) and *mouse* (small) flows. However, the performance on *rabbit* (middle-sized) flows decreases. As already stated in [5], this is due to the way the BMv2 queue



(a) Throughput

(b) Flow completion time

Fig. 4. Performance comparison on Spine-Leaf (4-4-20) topology

scheduler processes incoming frames, which does not guarantee that incoming frames at different ports of the switch are processed in order. This is a key issue for a correct execution of ARP-Path, which discovers paths based on latency and thus sees its performance reduced.

V. Conclusion

To the best of our knowledge, this paper presented the first use case of P4 registers to store stateful information and achieve autonomous forwarding in P4 pipelines of hybrid SDN devices. We used the P4 registers to implement the forwarding tables needed by ARP-Path, a distributed autonomous forwarding protocol, and make them coexist with the standard P4 forwarding tables operating under the SDN control plane command. By relying on standard P4 elements, we not only ensure the portability of the code to any P4 target, but also improve performance with respect to previous non-standard P4 implementations. However, the forwarding information derived autonomously by the hybrid SDN switches is still not readable by the centralised control plane, since the current P4/P4-Runtime specifications do not permit it. Hence, coexistence of both considered forwarding mechanisms is achieved by prioritisation techniques instead of implementing fully-cooperative mechanisms. To make fully-cooperative mechanisms become a reality, either access to P4 match-action tables from the data plane and/or control plane access to P4 registers should be granted in future revisions of the P4 specifications.

References

[1] F. Hu *et al.*, "A Survey on Software-Defined Network and OpenFlow: from Concept to Implementation," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 4, pp. 2181–2206, 2014.

[2] P. Bosshart *et al.*, "P4: Programming Protocol-independent Packet Processors," *SIGCOMM Computer Communication Review*, vol. 44, no. 3, pp. 87–95, 2014.

[3] K. Zheng *et al.*, "LazyCtrl: A Scalable Hybrid Network Control Plane Design for Cloud Data Centers," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 1, pp. 115–127, 2017.

[4] J. Alvarez-Horcajo *et al.*, "New Cooperative Mechanisms for Software Defined Networks based on Hybrid Switches," *Transactions on Emerging Telecommunications Technologies*, vol. 28, no. 8, p. e3150, 2017.

[5] I. Martinez-Yelmo *et al.*, "ARP-P4: Deep Analysis of a Hybrid SDN ARP-Path/P4Runtime Switch," *Telecommunication Systems*, p. 555–565, 2019.

[6] O. Awobuluyi, "Periodic Control Update Overheads in OpenFlow-Based Enterprise Networks," in *International Conference on Advanced Information Networking and Applications (AINA)*, 2014.

[7] S. Bhandarkar *et al.*, "Scalability Issues in Software Defined Network (SDN): A Survey," *Advances in Computer Science and Information Technology (ACSIT)*, vol. 2, no. 1, pp. 81–85, 2015.

[8] A. Hakiri *et al.*, "Software-Defined Networking: Challenges and Research Opportunities for Future Internet," *Computer Networks*, vol. 75, Part A, pp. 453 – 471, 2014.

[9] S. Hassas Yeganeh *et al.*, "Kandoo: A Framework for Efficient and Scalable Offloading of Control Applications," in *Workshop on Hot Topics in Software Defined Networks (HotSDN)*, 2012.

[10] P4 Consortium, "Behavioral Model: Rewrite of the Behavioral Model as a C++ project," 2018.

[11] N. Zilberman *et al.*, "NetFPGA SUME: Toward 100 Gbps as Research Commodity," *IEEE Micro*, vol. 34, no. 5, pp. 32–41, 2014.

[12] J. S. da Silva *et al.*, "Extern Objects in P4: an ROHC Header Compression Scheme Case Study," in *IEEE Conference on Network Softwarization and Workshops (NetSoft)*, 2018.

[13] Yu-Kuen Lai *et al.*, "Sketch-based Entropy Estimation for Network Traffic Analysis using Programmable Data Plane ASICs," in *2019 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*, 2019.

[14] D. Ding *et al.*, "Estimating Logarithmic and Exponential Functions to Track Network Traffic Entropy in P4," in *IEEE/IFIP Network Operations and Management Symposium (NOMS)*, 2019.

[15] E. Rojas *et al.*, "All-Path Bridging: Path Exploration Protocols for Data Center and Campus Networks," *Computer Networks*, vol. 79, pp. 120–132, 2015.

[16] J. Neruda, "Configuration of Remote P4 Device," in *EXCEL@FIT*, 2018.

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

[18] M. Alizadeh *et al.*, "CONGA: Distributed Congestion-aware Load Balancing for Datacenters," *SIGCOMM Computer Communication Review*, vol. 44, pp. 503–514, 2014.

[19] A. Greenberg *et al.*, "VL2: A Scalable and Flexible Data Center Network," *SIGCOMM Computer Commununication Review*, vol. 39, no. 4, pp. 51–62, 2009.

[20] M. Alizadeh *et al.*, "Data center TCP (DCTCP)," *SIGCOMM Computer Communication Review*, vol. 41, no. 4, pp. 63–74, 2010.