2018 IEEE 26th International Conference on Network Protocols

ARP-P4: A hybrid ARP-Path/P4Runtime switch

Isaias Martinez-Yelmo, Joaquin Alvarez-Horcajo, Miguel Briso-Montiano, Diego Lopez-Pajares, Elisa Rojas
University of Alcala. Alcala de Henares, Spain
{isaias.martinezy, j.alvarez, m.briso, diego.lopezp, elisa.rojas}@uah.es

***Abstract*—This paper presents ARP-P4, a hybrid switch based on a local ARP-Path control plane written in P4, but maintaining legacy P4 Runtime control plane capabilities. Its main purpose is to study the readiness of P4 to perform directly local control actions at the data plane, such as the required ones for the ARP-Path protocol. Thus, ARP-P4 defines a data plane that forwards any ingress packet locally via ARP-Path or remotely via P4 Runtime installed rules through an SDN controller. Finally, we discuss the current limitations of P4 non-standard externs to interact with P4 Runtime to develop autonomous capabilities.**

***Index Terms*—P4, P4 Runtime, SDN, Hybrid SDN, BMv2**

I. Introduction

In recent years, the Software-Defined Networking (SDN) paradigm has been crowned as the master key towards communication networks of the future [1]. Its founding protocol –OpenFlow [2]– was initially designed as a proof-of-concept and, although it has evolved together with SDN, it is unable to cope with the strict demands of new stakeholders [3], such as data plane programmability. The P4 language and the P4-Runtime specification [4] lately appeared to provide some of these desired capabilities unfulfilled by OpenFlow. Together with the development of these protocols, some authors defend that the SDN data plane should additionally convey part of the intelligence of the network to guarantee further possibilities. This concept –also denominated as hybrid SDN [5]– is being studied by the research community [6]. This article describes ARP-P4, a hybrid ARP-Path/P4-Runtime switch based on the P4 language and the Behavioral Model v2 (BMv2) target. Its objective is to explore the capabilities of the P4 specifications to allow autonomous behaviours on network devices. We have found that important restrictions exist as current specifications are focused in a fully centralised SDN control plane.

II. ARP-P4

The ARP-P4 design and development is based on the BMv2 [7] target, selected as it is specifically designed for prototyping. ARP-Path switches are autonomous, they do not depend on any controller to establish paths, but P4 is designed to define data planes assuming an external control plane. Thus, it is necessary to define how our ARP-P4 switch can establish paths without depending on any other control entity. ARP-P4 uses ARP-Path [8] to obtain these paths by selecting the fastest path between a source and a destination via an exploration mechanism based on controlled flooding. Furthermore, it would be desirable to insert forwarding rules via P4 Runtime if possible.

P4 is primarily designed to perform the ingress packet processing via P4 tables, registers, actions, etc. from an external controller. However, ARP-P4 must be also able to forward packets autonomously with ARP-Path when no matching rules exist for them. This requires the deployment and management of a MAC address Learning Table (LT), as in traditional switches. As the LT is updated according to ingress packets, the most convenient approach would be to define the LT on P4 and later manage its table entries according the ARP-Path protocol to define the forwarding rules. A straightforward alternative considered by the P4 specification is the use of a local controller with P4 Runtime support on each switch. However, this option would require a local controller per switch, which is not always feasible. Thus, we advocate to use the P4 non-standard *extern* methods to establish the egress port for ingress packets and update the LT when necessary. The intention was to explore the current status of non-standard extern objects in the P4 language and evaluate how its use can contribute to the definition of P4-based local control planes.

This extern object is managed by an extern interface with several methods for this purpose. However, a drawback of this proposal resides on the unawareness of the ARP-Path forwarding state by external control planes (e.g. SDN controllers), due to the unavailable interoperability of non-standard extern objects with P4 Runtime. Therefore, we defined an ARP-Path extern interface that uses the extern object that represents ARP-Path's LT. The LT stores the learnt MAC addresses with their egress port together with a configurable timestamp (defined in ARP-Path to prevent loops). The beginning of the Fig. 2 shows the definition of the P4 ARP-Path extern interface with its methods and their description. Aditionally, any P4-based switch needs a pipeline to process the packets. After the definition of the ARP-Path extern API, Fig. 2 shows a simple P4 pipeline that forwards ingress packets using the ARP-Path extern API. ARP-P4 integrates this pipeline with the regular P4Runtime control plane and prioritises the P4Runtime forwarding over ARP-Path forwarding. Hence, the coexistence between a local (ARP-Path) and a remote (SDN controller) control plane is achieved (specfically, we tested it with ONOS).

III. Evaluation

We compare the performance of ARP-P4, ARP-Path [6] and ECMP in terms of throughput (average Mbps per flow) under different network loads and flow traffic distributions, following the procedure in [9]. The results are shown in Fig. 1, divided

This work was funded by a grant from the Comunidad Autónoma de Madrid through Project TIGRE5-CM S2013/ICE-2919, and by the University of Alcalá through programs "Formación del Profesorado Universitario" and "Ayuda de Iniciación en la Actividad Investigadora".

978-1-5386-6043-0/18/$31.00 ©2018 IEEE
DOI 10.1109/ICNP.2018.00062

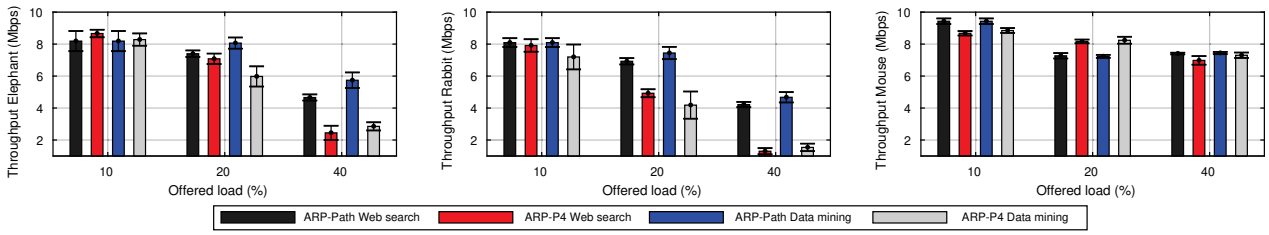

Fig. 1: Throughput on Spine-Leaf topology.

```
extern ArpPath {
  ArpPath(bit<64> size);
/* Broadcast traffic and ARP Request processing (LT) */
    void flood(in bit<48> addr, in bit<9> port,
                                in bit<16> ethType);
/* Unicast traffic forwarding (according to LT)*/
    void forward(in bit<48> addr);
/* ARP Reply processing (LT and timestamp update)*/
    void path_reply(in bit<48> addr, in bit<9> port);
/* Statistics collection from the LT */
    void num_entr(out bit<32> n);
}
control ArpPathPipeline(inout headers_t hdr,
    inout standard_metadata_t metadata,
    ArpPath extern_arp_path) {
  apply{
    if (hdr.ethernet.dstAddr == BROADCAST)
      extern_arp_path.flood(hdr.ethernet.srcAddr,
         metadata.ingress_port, hdr.ethernet.etherType);
    else{
      extern_arp_path.forward(hdr.ethernet.dstAddr);
      if (hdr.ethernet.etherType==ARP_TYPE){
        extern_arp_path.path_reply(hdr.ethernet.srcAddr,
                                   metadata.ingress_port);
} } } }
```

Fig. 2: ARP-P4 Control Block source code

into three graphs: one per flow type. As expected, throughput decreases with offered load regardless of the protocol and traffic distribution. When the number of flows contending for the same resources (link capacity) increases, each flow obtains a smaller share of the available resources. Furthermore, they spend longer time in the network due to the smaller obtained throughput, which aggravates this effect. ECMP and ARP-Path have similar performance, but ARP-Path needs no controller.

ARP-P4 should obtain the same performance than ARP-Path, which is proved to be true while offered load is small but not when it is high (40% of offered load in Fig. 1). This effect is specially noticeable on elephant and rabbit flows (irrespective of the flow size), which are the main contributors to the offered load in the conducted experiments. We explored the internals of the BMv2 target to find possible explanations to this unexpected low performance, and we found out that BMv2 does not process ingress packets in the same arrival order. BMv2 dequeues one packet from each ingress port following a Round Robin (RR) policy. Hence, the arrival time is not maintained to later process the ingress packets, which hides the actual delays of the discovered paths from the the controlled flooding implemented by the flood extern method. This fact has a negative impact on ARP-P4 since the fastest paths (less congestioned) are not properly discovered, specially on high loads with lots of packets in the ingress queues.

IV. Conclusions

To the best of our knowledge, ARP-P4 is the first autonomous BMv2-based switch developed using the P4 language. ARP-P4 is implemented in a BMv2 target together with a P4 program that makes use of non-standard extern methods, with no local controller, to forward packets using the ARP-Path rules. However, P4 Runtime has no access to the forwarding rules set by the ARP-Path extern object, neither the ARP-Path extern API can interact with P4 Runtime with the current P4 specifications. This undesirable side effect should be resolved in future P4 specifications. In any case, rules established through P4 Runtime from an external controller are properly installed allowing a fully featured hybrid switch. Finally, the ARP-P4 performance results on high loads are below the expectations since BMv2-based targets do not maintain the arrival time of the packets. We expect that the findings in this paper will contribute to improve the P4 language and P4 Runtime specifications by also allowing the definition of local control planes exclusively based on P4.

References

[1] D. Kreutz, F. M. V. Ramos, P. E. Verssimo, 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, Jan 2015.

[2] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: Enabling innovation in campus networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, Mar. 2008.

[3] E. Rojas, "From Software-Defined to Human-Defined Networking: Challenges and Opportunities," *IEEE Network*, vol. 32, no. 1, pp. 179–185, Jan 2018.

[4] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker, "P4: Programming Protocol-independent Packet Processors," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 3, pp. 87–95, Jul. 2014.

[5] R. Amin, M. Reisslein, and N. Shah, "Hybrid SDN Networks: A Survey of Existing Approaches," *IEEE Communications Surveys Tutorials*, pp. 1–1, 2018.

[6] J. Alvarez-Horcajo, I. Martinez-Yelmo, E. Rojas, J. A. Carral, and D. Lopez-Pajares, "New cooperative mechanisms for software defined networks based on hybrid switches," *Transactions on Emerging Telecommunications Technologies*, vol. 28, no. 8, p. e3150, 2017, e3150 ett.3150.

[7] "bmv2: Designing your own switch target with bmv2." [Online]. Available: http://www.bmv2.org

[8] G. Ibanez, J. A. Carral, J. M. Arco, D. Rivera, and A. Montalvo, "ARP-Path: ARP-Based, Shortest Path Bridges," *IEEE Communications Letters*, vol. 15, no. 7, pp. 770–772, July 2011.

[9] J. Alvarez-Horcajo, D. Lopez-Pajares, J. M. Arco, J. A. Carral, and I. Martinez-Yelmo, "Tcp-path: Improving load balance by network exploration," in *2017 IEEE 6th International Conference on Cloud Networking (CloudNet)*, Sept 2017, pp. 1–6.