Deploying PolKA Source Routing in P4 Switches

(Invited Paper)

Cristina Dominicini, Rafael Guimarães, Diego Mafioletti, Magnos Martinello, Moises R. N. Ribeiro, Rodolfo Villaça, Frédéric Loui, Jordi Ortiz, Frank Slyne, Marco Ruffini, and Eoin Kenny

***Abstract*—One great challenge of modern networks is how to select paths and react to highly variable and demanding traffic patterns. In recent years, emerging networking languages and architectures, such as P4/PISA, have created unprecedented opportunities for rapidly prototyping disruptive solutions in programmable data planes. In this direction, source routing (SR) is a prominent alternative to conventional table-based routing, since it reduces the convergence time of building distributed routing tables. In particular, PolKA explores the polynomial residue number system (RNS) for a fully stateless SR, i.e., no state update on packet headers needs to be conveyed along the route. This paper reports the deployment of PolKA in a continental testbed composed of P4 programmable switches. Results for end-to-end throughput and forwarding latency show that PolKA matches the data plane performance of traditional approaches, while paving the way for further innovative applications by exploring RNS properties in dynamic scenarios.**

I. Introduction

In recent years, the advances in software defined networking (SDN) have enabled the emergence of programmable network devices that allow the packet processing behaviour in the data plane to be reconfigured on the fly in a systematic fashion [1]. This unleashed the development of custom protocols for the next generation of network applications.

To ensure maximum performance and resiliency of these network applications, traffic engineering should be able to select among all possible paths in the underlay network according to dynamic demands [2]. However, traditional routing approaches are limited by the size of forwarding tables and restrict the number of paths that can be represented. Moreover, as modern traffic engineering is based on logically centralised decision making for path selection (e.g., MPLS and SDN), novel routing proposals have the potential to reduce the management burden of building up distributed routing tables.

In this context, the source routing (SR) paradigm has been gaining momentum [3], [2], [4]. In SR, the responsibility of defining the route belongs to the source of packets (or edge nodes), which can specify all the elements of the path. Then, the route can be inserted in the packet header, and used by each node to define the output port, without executing table lookup or communicating with controllers. Thus, it reduces control signalling and latency for path setup as well as exploits all existing paths to achieve maximum throughput [2].

Cristina Dominicini, Rafael Guimarães, Diego Mafioletti are with Federal Institute of Espírito Santo; Magnos Martinello, Moises Ribeiro, and Rodolfo Villaça are with Federal University of Espírito Santo; Frédéric Loui is with RENATER; Jordi Ortiz is with University of Murcia; Frank Slyne and Marco Ruffini are with Trinity College Dublin; and Eoin Kenny is with HEAnet.

In particular, some SR solutions explore the Residue Number System (RNS) [5], where the output port is given by the remainder of the division (i.e., a *modulo* operation) of a route label by the node identifier [5]. A fundamental property of this approach is a fully stateless forwarding without any header modification [6]. However, the *modulo* operation with non-constant operands is not supported by current programmable switches. To solve this problem, in a previous work, the authors proposed PolKA [6], a RNS-based SR scheme that replaces the integer arithmetic used by the original scheme [5] with the binary polynomial arithmetic (Galois field (GF) of order 2 or GF(2) [7]). The immediate benefit was to enable the reuse of commodity network functions based on polynomial arithmetic. In fact, PolKA proposed to explore the Cyclic Redundancy Check (CRC) hardware to execute the *modulo*, and evaluated it in an emulated environment.

In this paper, we propose an implementation that suits commercial programmable switches to prove PolKA's effectiveness in a real-world environment. The contributions are: (i) we implement PolKA using the P4 language in the high-performance switching ASIC Tofino [8]; (ii) we discuss the potential applications and some deployment guidelines; and (iii) we deploy our implementation in the continental RARE/GÉANT P4 Lab testbed [9] and conduct the first hardware-based comparison of PolKA with traditional approaches.

This paper is structured as follows. Section II discusses the background, the potential applications, and the deployment guidelines, followed by the evaluation of the prototype in Section III. Finally, Section IV discusses our conclusions.

II. PolKA SR

PolKA [6] is a RNS-based SR scheme that explores the Chinese remainder theorem (CRT) for polynomials [7]. As shown in Fig. 1, the architecture is composed of: (i) edge nodes, (ii) core nodes, and (iii) an SDN Controller, responsible for configuring the nodes. The SR relies on three polynomials over GF(2): (i) *nodeID*: a fixed identifier assigned to core nodes by the Controller in a network configuration phase; (ii) *portID*: an identifier assigned to the output ports of each core node; and (iii) *routeID*: a route identifier, calculated by the Controller and embedded into the packets by the edge nodes.

Let S = $\{s_1(t), s_2(t), \ldots, s_N(t)\}$ be the multiset of the irreducible polynomials representing the *nodeID*s of the nodes in a given path, and O = $\{o_1(t), o_2(t), \ldots, o_N(t)\}$ be the multiset of N polynomials, where $o_i(t)$ represents the output port for the packet at the core node $s_i(t)$, for $i = 1, 2, \ldots N$,

978-3-903176-33-1

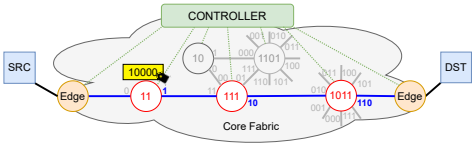


Fig. 1. Example of PolKA SR.

satisfying the condition that $deg(s_i) > deg(o_i)$. For instance, if $o_i(t) = 1 \cdot t^2 + 1 \cdot t$, it maps the port 110 (or label 6).

Using the polynomial CRT, the Controller calculates the *routeID* as the polynomial $R(t)$ that satisfies the condition $R(t) \equiv o_i(t) \ (mod \ s_i(t))$, for $i = 1, 2, ..., N$, where [6]: $R(t) =< \tilde{R}(t) >_{M(t)}$, $\tilde{R}(t) = \sum_{i=1}^{N} o_i(t) \cdot m_i(t) \cdot n_i(t)$, $M(t) = \prod_{i=1}^{N} s_i(t)$, $n_i(t) \cdot m_i(t) \equiv 1 \quad (mod \quad s_i(t))$, and $m_i(t) = M(t)/s_i(t)$. The Controller may proactively compute $R(t)$ or calculate it when the first packet of a flow arrives, with an algorithm complexity of $\mathcal{O}(len(M)^2)$ [7]. On the other hand, core nodes only execute a remainder of the euclidean division of the *routeID* by its *nodeID*: $o_i(t) = < R(t) >_{s_i(t)}$

Fig. 1 shows an example of PolKA SR for a path composed of three nodes, which received their *nodeID*s from the Controller: $s_1(t) = t + 1$ (or 11), $s_2(t) = t^2 + t + 1$ (or 111), $s_3(t) = t^3 + t + 1$ (or 1011). For this path, the *portID*s are: $o_1(t) = 1$, $o_2(t) = t$ (or 10), $o_3(t) = t^2 + t$ (or 110). The *routeID* calculated by the Controller is $R(t) = t^4$ (or 10000) [6]. Thus, each node can calculate the *portID* using a *modulo* operation. For example: at s_3, $o_3(t) = < 10000 >_{1011} = 110$.

A. Properties and Applications

SR presents several advantages over table-based methods, such as: fast path configuration, optimised network usage, smaller control overhead, and reduced TCAM usage [2]. List-based SR (LSR), a traditional way for executing SR, represents the *routeID* as a list of ports or addresses and the forwarding operation as a pop [3], [4]. Nevertheless, RNS-based SR presents additional properties, not present in LSR, such as:

1) Forwarding without header modification: The forwarding is the direct result of the *modulo* operation. In contrast, LSR rewrites the route label to update the position in the list.

2) The node order is irrelevant to derive the routeID: In the CRT, data from each node (*nodeID* and *portID*) belongs to its own addend of the summation and does not influence the other summation addends, as the finite summation is commutative.

3) The path information is not transmitted in the clear: To derive the output port, it is necessary to know both the *routeID* of the packet and the *nodeID* of the node.

Therefore, RNS-based SR can explore these properties to provide innovative networking functionalities, such as: (i) convergent switches in scenarios where header modification can be hardly implementable (e.g., all-optical switches) [5]; (ii) intrinsic fast failure recovery by adding redundant nodes in the *routeID* [10], [11]; (iii) agile path reconfiguration [12], and traffic differentiation with Quality of Service (QoS) [13]; (iv) simultaneous operation with table-based forwarding in a hybrid approach to save TCAM for specific flows; (v) multi-domain Service Function Chaining (SFC), by using gateways that rewrite the SFC labels across domains [12]; (vi) multipath routing by extending the original scheme, if the coefficients of $o_i(t)$ are used to represent the transmitting state of the ports; (vii) hardware synthesis with reduced costs and Network on Chip by exploring the polynomial arithmetic in hardware description languages [14]; (viii) systems based on Chaum's Mix-Net design [15] could be implemented by PolKA nodes to support onion routing and anonymity; (ix) since the header does not change throughout the path, the source can sign the *routeID* to provide route authenticity; and (x) modern homomorphic encryption systems with RNS [16] by using PolKA nodes to natively implement RNS operations.

B. Data Plane Deployment

In our implementation, we inserted the PolKA header between the Ethernet and IP headers. For simple SR, the header includes only a *routeID* field, but it can be extended to include extra fields for QoS [13], SFC [12] and resilience [11]. The length of these headers depend on [6], [12]: the number of nodes, the number of ports, the network diameter, the number of chain segments, and the number of traffic classes. PolKA can be deployed in P4 switches [8], and explores a technique that executes the polynomial *modulo* by using two *SHIFT*, one CRC, and two *XOR* operations [6]. The Tofino Native Architecture (TNA) supports CRC operations with custom polynomials of degree up to 32 [8].

C. Control Plane Deployment

The Controller assigns unique *nodeID*s to each core node in a network configuration phase. It calculates the irreducible polynomials for the *nodeID*s, ensuring they support the number of ports: $(deg(s_i) \geqslant \lceil \log_2(nports) \rceil)$. When a new flow reaches an edge switch (or proactively), the Controller calculates the *routeID* to define an end-to-end path across the core network. Then, the Controller installs a new table entry in the edge switch, which adds the *routeID* in the packet header. For the RNS computation with GF(2) operations, we developed and published a Python library[1] to be installed via `pip` tool.

III. Evaluation

The proof-of-concept (PoC) aims to prove that PolKA can be: (i) deployed in commercial P4 programmable switches; and (ii) implemented with similar performance to traditional table-based and LSR approaches. To this end, we deployed the forwarding methods using the P4-16 language in the RARE/GÉANT P4 Lab [9]. The PoC comprises four Intel/ Barefoot Tofino WEDGE100BF32X switches that are geographically distributed and connected by a 10Gbps optical link (see Fig. 2): Amsterdam (S_4 AMS), Frankfurt (S_1 FRA), Budapest (S_2 BUD), and Poznan (S_3 POZ). Two edge nodes generate traffic using the `pktgen-dpdk` tool version 19.12 (DPDK 19.11.5), capable of replaying packets from *pcap* files. The tests explored the longest path (S_1-S_2-S_3-S_4), and Table I details the configurations, as explained below.

[1]https://pypi.org/project/polka-routing/

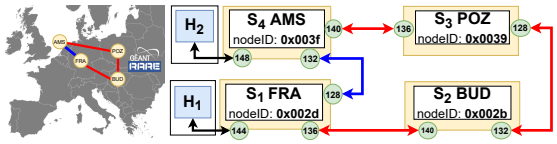
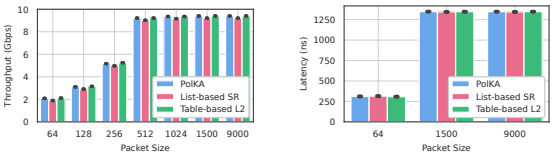


Fig. 2. RARE/GÉANT P4 Lab European testbed [9].

TABLE I
END-TO-END CONFIGURATION.

Path	PolKA Key	List-based SR bos / port	Table-based L2 Switch / Match / Port
H_1-S_1-S_2-S_3-S_4-H_2	0x583585abfe73a523	0 / 136 0 / 132 0 / 136 1 / 148	S_1 / 01:01:01:00:00:02 / 136 S_2 / 01:01:01:00:00:02 / 132 S_3 / 01:01:01:00:00:02 / 136 S_4 / 01:01:01:00:00:02 / 148
H_2-S_4-S_3-S_2-S_1-H_1	0x6b06b6a3544c62a6	0 / 140 0 / 128 0 / 140 1 / 144	S_4 / 01:01:01:00:00:01 / 140 S_3 / 01:01:01:00:00:01 / 128 S_2 / 01:01:01:00:00:01 / 140 S_1 / 01:01:01:00:00:01 / 144



(a) Throughput. (b) Forwarding latency.

Fig. 3. Comparison PolKA vs. List-based SR vs. Table-based L2.

*1) **PolKA**:* Table I presents the *routeID*s for the longest path. In our PoC, we adopted polynomials of degree 16, but it is possible to use smaller polynomials. For this network diameter (4 hops), the size of the PolKA header was 64 bits.

*2) **LSR**:* We defined the header as: 1 bit for the bos (bottom of stack) and 15 bits for the port number. Hence, for 4 hops, the array of headers has 64 bits. In LSR, the output port is directly available in the SR header. For the longest path, the array is: $[(0/136), (0/132), (0/136), (1/148)]$.

*3) **Table-based L2**:* We added 2000 random table entries in the switches, including the entries to provide communication between H_1 and H_2 for the longest path. Table I shows the table entries to enable communication from H_1 (MAC address 01:01:01:00:00:01) to H_2 (MAC address 01:01:01:00:00:02).

A. End-to-end throughput comparison

Fig. 3(a) shows the throughput comparison at H_2 for different packet sizes: (i) the throughput grows linearly with the increase of the packet size in all forwarding methods, but when the packet size is lower than 512 bytes, the NIC of the edge node cannot provide line rate; and (ii) the average throughput with its standard deviation is small, and they are in the same order of magnitude for all methods for the different packet sizes. Finally, PolKA supports high throughput (10Gbps) and high packets per second rates (more than $2.5 * 10^6$).

B. Forwarding latency comparison

We compared the forwarding latency in the P4 pipeline at the BUD node by exploring two 48-bit hardware timestamps from the Tofino's ASIC: (i) ingress global: timestamp (ns) taken upon arrival at ingress; and (ii) egress global: timestamp (ns) taken upon arrival at egress. To this end, the P4 code adds these timestamps in the IPv4 options field when the packet arrives at the ingress and egress control blocks. In this test, we captured 10000 packets at the destination, and parsed them offline to extract the forwarding latency by subtracting the egress global timestamp from the ingress global timestamp. We used the `pktgen-dpdk` tool to generate UDP traffic for different packet sizes with a throughput of 10Gbps. As shown in Fig. 3(b), the average for 64-byte packets is about 300ns, whereas, for 1500-byte and 9000-byte packets, the average is about 1300ns, as expected in a store-and-forward mode.

IV. CONCLUSIONS

In this work, we described PolKA's implementation in a high-performance P4 switching ASIC Tofino and deployment at the continental RARE/GÉANT P4 Lab [9]. Experimental results have shown that PolKA matches the performance of traditional L2 table-based forwarding and LSR approaches. As future works, we plan to explore RNS properties to enable innovative networking applications, and propose deployment guidelines for production use cases in dynamic scenarios.

ACKNOWLEDGMENT

This work was funded by FAPES, CAPES (Programa de Desenvolvimento da Pós-Graduação - PDPG - Parcerias Estratégicas nos Estados), and CNPq. We thank the GÉANT P4 Lab, which is funded by the GÉANT Grant Agreement No. 856726 (GN4-3), the Intel Connectivity Research Program, and Vladimir Gurevich for the contributions.

REFERENCES

[1] R. Bifulco and G. Rétvári, "A survey on the programmable data plane: Abstractions, architectures, and open problems," in *IEEE International Conference on High Performance Switching and Routing*, 2018, pp. 1–7.
[2] S. A. Jyothi *et al.*, "Towards a flexible data center fabric with source routing," in *ACM SIGCOMM SOSR*. ACM, 2015, pp. 10:1–10:8.
[3] C. Filsfils *et al.*, "The Segment Routing Architecture," in *IEEE Global Communications Conference (GLOBECOM)*. IEEE, 2015, pp. 1–6.
[4] X. Jin *et al.*, "Your data center switch is trying too hard," in *ACM SIGCOMM SOSR*. ACM, 2016, pp. 12:1–12:6.
[5] M. Martinello *et al.*, "KeyFlow: a prototype for evolving SDN toward core network fabrics," *IEEE Network*, vol. 28, no. 2, pp. 12–19, 2014.
[6] C. Dominicini *et al.*, "Polka: Polynomial key-based architecture for source routing in network fabrics," in *IEEE NetSoft*, 2020, pp. 326–334.
[7] V. Shoup, *A computational introduction to number theory and algebra*. Cambridge university press, 2009.
[8] Intel, "Open Tofino." [Online]. Available: https://github.com/barefootnetworks/Open-Tofino
[9] GÉANT, https://wiki.geant.org/pages/viewpage.action?pageId=148085131, 2021, (Accessed on 02/03/2021).
[10] R. R. Gomes *et al.*, "KAR: Key-for-any-route, a resilient routing system," in *IEEE/IFIP DSN Workshop*, June 2016, pp. 120–127.
[11] A. Liberato *et al.*, "Rdna: Residue-defined networking architecture enabling ultra-reliable low-latency datacenters," *IEEE Transactions on Network and Service Management*, vol. 15, no. 4, pp. 1473–1487, 2018.
[12] C. K. Dominicini *et al.*, "KeySFC: Traffic steering using strict source routing for dynamic and efficient network orchestration," *Computer Networks*, vol. 167, p. 106975, 2020.
[13] W. Froes *et al.*, "Proglab: Programmable labels for qos provisioning on software defined networks," *Computer Communications*, vol. 161, pp. 99–108, 2020.
[14] M. Ruaro *et al.*, "Software-defined networking architecture for noc-based many-cores," in *2018 IEEE ISCAS*, May 2018, pp. 1–5.
[15] D. L. Chaum, "Untraceable electronic mail, return addresses, and digital pseudonyms," *Commun. ACM*, vol. 24, no. 2, p. 84–90, Feb. 1981.
[16] M. Gomathisankaran *et al.*, "Horns: A homomorphic encryption scheme for cloud computing using residue number system," in *2011 45th Annual Conference on Information Sciences and Systems*, 2011, pp. 1–5.