A lifecycle experience of PolKA: From prototyping to deployment at Géant Lab with RARE/FreeRouter

Everson S. Borges[1,2], **Edgard Pontes**[1], **Cristina Dominicini**[2], **Marcos Schwarz**[3], **Csaba Mate**[4], **Frederic Loui**[5], **Rafael Guimarães**[2], **Magnos Martinello**[1], **Rodolfo Villaça**[1], **Moisés R. N. Ribeiro**[1]

[1]Federal University of Espírito Santo (UFES)
Av. Fernando Ferrari, 514, Goiabeiras – 29.075-910 – Vitória – ES – Brazil

[2]Federal Institute of Espírito Santo (IFES)

[3]RNP-National Research Network

[4]RARE - Router for Academia Research & Education

[5]GÉANT - Pan-European Research and Education Networks

{everson.borges,edgard.pontes}@edu.ufes.br

Abstract. *In this paper, we take the position of developers that need to deploy emerging programmable protocols (or services) with specific network requirements and want to know how to benefit from open router platform and testbed infrastructure. Our focus is to exploit PolKA lifecycle experience as a use case to figure out a balance on integrating and reusing legacy protocols with new protocols.*

Resumo. *Neste artigo, assumimos a posição de desenvolvedores que precisam implantar protocolos (ou serviços) programáveis emergentes com requisitos de rede específicos e querem saber como se beneficiar da plataforma de roteador aberta e da infraestrutura de teste. Nosso foco é explorar o equilíbrio na integração e reutilização de protocolos legados com novos protocolos.*

1. Introduction

Source routing (SR) is a prominent alternative to table-based routing for reducing the number of network states. In this approach, a source adds a route label in the packet header to define the paths along the network. Segment Routing and PolKA [Dominicini et al. 2020] are two new approaches of source routing paradigm that allows a flow to be routed to a specific topological path, while maintaining per-flow state only at the ingress node to the SR domain.

However, a big challenge to develop a new protocol is how to design and implement it with its own encapsulation representing an encoding abstraction that needs to be efficiently processed by the switches. For example, to implement an integer modulo operation that is not natively supported in commodity or even programmable network hardware. Therefore, a network developer needs to use software switches [Martinello et al. 2014], or depended on synthesizing integer division to ASICs or NetFPGAs [Liberato et al. 2018].

In this paper, we take the position of developers that need to deploy emerging programmable protocols (e.g. PolKA[Dominicini et al. 2020]) with specific network requirements (e.g. mod operation) and want to know how to benefit from open router platform and testbed infrastructure. Our focus is to exploit the balance on integrating and reusing legacy protocols with new protocols.

2. PolKA and FreeRouter in a nutshell

2.1. PolKA- Polynomial Key-Based Architecture

Let us assume that a packet should be routed via a selected path, represented by N core nodes and their respective output ports. Let S = $\{s_1(t), s_2(t), \ldots, s_N(t)\}$ be the multiset of the polynomials representing the *nodeIDs* of the nodes in this path. The set S must be composed of pairwise co-prime polynomials, and satisfy the condition $degree(s_i(t)) \geq \log_2(nports)$, where $nports$ denotes the number of ports in the node. For simplicity, we assume that $s_i(t)$ are irreducible polynomials. Let 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 $i = 1, 2, \ldots N$, satisfying the condition that $degree(s_i) > degree(o_i)$. For instance, if the output port polynomial is $o_i(t) = 1 \cdot t^2 + 1 \cdot t$, it maps the port 110 and the packet is forwarded to port label 6 at node $s_i(t)$. Based on the definition of the path represented by S and O, the Controller calculates the *routeID* using the polynomial CRT [Shoup 2009, Bajard 2007] as the polynomial $R(t)$ that satisfies:

$$R(t) \equiv o_i(t) s_i(t), \quad for \quad i = 1, 2, \ldots N \tag{1}$$

The *routeID* is embedded in the packet by the edge, and the forwarding operation in each core node calculates the output port as the remainder of the euclidean division of the *routeID* in the packet by its *nodeID*: $o_i(t)$ = $< R(t) >_{s_i(t)}$

Fig. 1 shows an example for a path composed of three core nodes, which received their *nodeIDs* from the Controller in a network configuration phase: $s_1(t) = t + 1 = 11, s_2(t) = t^2 + t + 1 = 111, s_3(t) = t^3 + t + 1 = 1011$. Considering the path $s_1 \rightarrow s_2 \rightarrow s_3$, the set O is composed by: $o_1(t) = 1, o_2(t) = t = 10, o_3(t) = t^2 + t = 110$. For this example, the *routeID*, calculated according to the polynomial CRT, is $R(t) = 10000$. The Controller may proactively compute this $R(t)$ or calculate it when the first packet of a flow arrives. To configure the path, the Controller installs flow entries in the edges, which embed the *routeID* 10000 into the packets. Then, each node can calculate its *portID* by dividing the *routeID* of the packet by its own *nodeID*. For example, the remainder of $R(t) = 10000$ divided by $s_2(t) = 111$ is $o_2(t) = 10$ (port label 2).

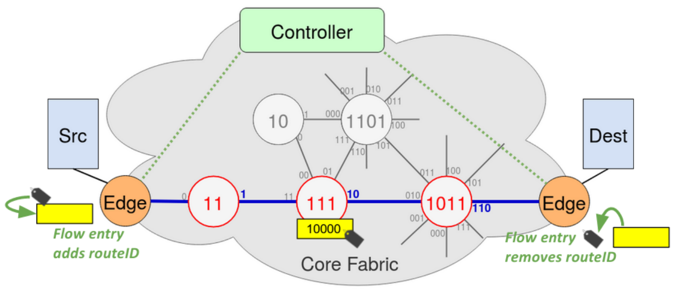


Figure 1. Example of PolKA SR.

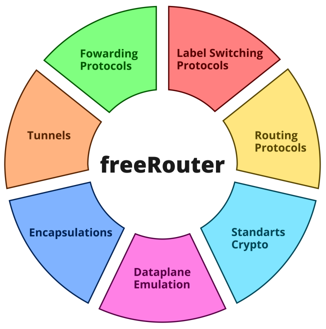


Figure 2. FreeRouter.

2.2. FreeRouter

FreeRouter is a control plane platform shown in **Fig. 2**, which uses UNIX socket to forward packets and this is a key feature that is leveraged to connect the control plane **FreeRouter** to any type of protocol plane legacy, such as OSPF, IPv4, IPV6, ATM, ethernet, etc. In **FreeRouter** everything is in a VRF so there is no global VRF, this design choice has positive consequences like. There are no VRF recognition questions, having multiple BGP processes for the same **FreeRouter** instance linked to a different VRF. in addition, **FreeRouter** uses dual stack, so the entire feature set supports both IPv4 and IPv6 without compromise.

In order to support legacy applications, **FreeRouter** brings along a densily populated control plane able to handle: **i)** diverse encapsulation; **ii)** cryptography; **iii)** both table-based and also lable-based at L2 and L3 forwarding; **iii)** diverse tunneling; **iv)** both IGP and EGP routing protocols; and **v)** Policy-Based Routing services, NAT, QoS, and other services, highlighted by **Fig. 2**. More detailed list of protocols on different layers (and their combinations) implemented and tested at the moment (on different data planes) are available in RARE project[1].

3. PolKA lifecycle experience

This section describes the PolKA lifecycle experience from its proposal, prototyping, validation, deployment, and integration in production grade testbeds.

3.1. Initial proposal: Prototyping PolKA in an emulated environment

Previous works explored the integration of RNS-based SR with SDN [Martinello et al. 2014, Liberato et al. 2018]. However, these works relied on integer RNS arithmetic, and the integer modulo operation cannot be implemented in current commodity network hardware. Therefore, they either used software switches implementations [Martinello et al. 2014], or depended on synthesizing integer division to ASICs or NetFPGAs [Liberato et al. 2018].

To solve this problem, PolKA was introduced [Dominicini et al. 2020], as a RNS-based SR scheme that replaces the integer arithmetic by polynomial arithmetic (Galois field (GF) of order 2 or GF(2)). The immediate benefit was to enable the reuse of commodity network functions based on polynomial arithmetic. In fact, PolKA explores the Cyclic Redundancy Check (CRC) hardware to execute the *modulo*, and evaluated it in an emulated environment with Mininet and P4-16 language. A hardware prototype with Netronome SmartNICs was used for measuring forwarding latency, but, as these SmartNICs only support fixed CRC polynomials, it restricts the use of its CRC hardware for PolKA.

3.2. PolKA as use case of the RARE testbed: Prototyping for Tofino targets

Then, PolKA was selected as an use case in RARE/GÉANT P4 Lab[2]. The partnership enabled an implementation of PolKA with the high-performance switching ASIC Tofino, and the first hardware-based comparison of PolKA with traditional approaches. As a result, we deployed a PoC [Dominicini et al. 2021], which comprised four Intel/Barefoot Tofino WEDGE100BF32X switches that were geographically distributed in Europe and connected at 10Gbps links. The experimental results showed that PolKA matches the data plane performance of traditional L2 table-based forwarding and list-based source routing.

In these experiments, the focus was on the data plane evaluation, and the control plane was implemented as a set of scripts that automated the essential node configuration tasks. In addition, our P4-based data plane was individually deployed in each switch.

[1]`http://rare.freertr.org`

[2]`https://wiki.geant.org/pages/viewpage.action?pageId=148085131`

3.3. Integration of PolKA in RARE/FreeRouter

The next step was the integration of **PolKA** in the RARE project. Besides being the first non-standard protocol to be readily available for any experimenter in RARE, we inherited various features from the **FreeRouter** platform, such as diverse encapsulations, access control lists, and policy-based routing. Moreover, **FreeRouter** allows the emulation of any topology with the same configuration files that can be tested and, then, deployed in a real P4-enabled infrastructure. In addition, **PolKA** can now be evaluated with baseline approaches, like Segment Routing, which are also supported by RARE/**FreeRouter**.

The control plane in the RARE project is based on **FreeRouter**. Thus, to develop PolKA's control plane we have to options: (i) to develop a centralized controller that interacts with the **FreeRouter** API, or (ii) to reuse standard distributed protocols already running in RARE and **FreeRouter**. As a first integration effort, we chose the second option, by integrating the PolKA control plane logic to the **FreeRouter** workflow [3] to: (i) retrieve available topology information from link-state routing protocols to calculate the routeIDs when creating **PolKA** tunnels, and (ii) reuse the Segment Routing indexes of the nodes to create a static table that maps these indexes to nodeID polynomials.

The DPDK framework and the P4 behavioural language are used to describe the packet processing behaviour of RARE data plane. The supported P4 target platforms are the BMv2 software switch with the V1Model architecture, and the Tofino high performance network processor with the PSA (Portable Switch Architecture) architecture. For the integration of **PolKA** data plane, we ported our previous code for both the DPDK and the Tofino data planes [4].

In this implementation, our fixed-length PolKA header can use different encapsulations (e.g., MPLS, Ethernet, VLAN, and PPP), and the routeID has 128 bits with CRC 16, which allows the maximum of 8 hops.

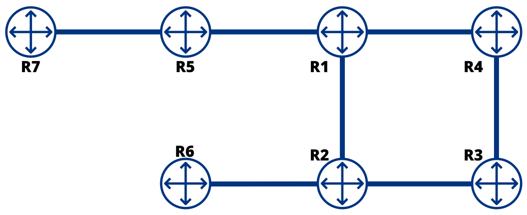


Figure 3. Edge-Core Experiment

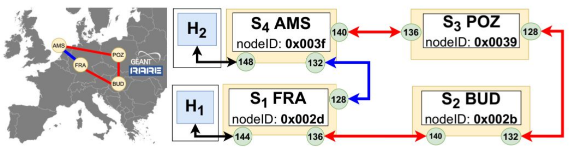


Figure 4. RARE/GEANT European testbed

4. Deployment and validation of PolKA with FreeRouter

As proof of concept, it was implemented a ring topology which R1, R2, R3 and R4 are core nodes, while R5 and R6 are edge nodes and R7 as host for running the experiments, as shown in the **Fig. 3**. This topology is composed by a shortest and a longest path chosen to be equivalent to the topology of RARE/Géant P4 Lab testbed (**Fig. 4**).

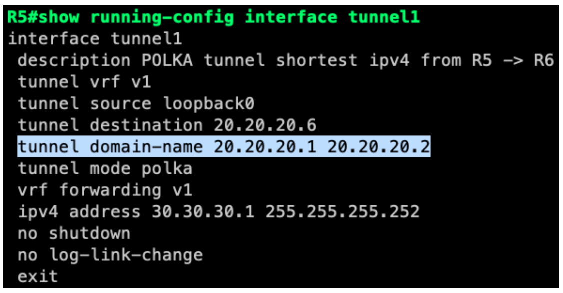
4.1. PolKA Experiment in FreeRouter

Retrieving topology information and configuring nodeIDs: For these functionalities, our design choice was to reuse legacy protocols rather than re-implementing these features

[3]`https://docs.freertr.org/guides/reference/`

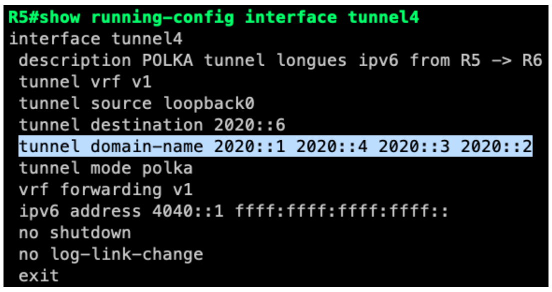
[4]`https://bitbucket.software.geant.org/projects/RARE/repos/rare/browse/p4src/include`

in a centralized controller. In particular, for this experiment, the OSPF routing protocol was enabled on all ethernet interfaces to gather the topology information. Besides, we map the already assigned unique Segment Routing indexes of each router to an index of a static table of irreducible polynomials to set a unique **PolKA**'s nodeID for each router.



```
R5#show running-config interface tunnel1
interface tunnel1
 description POLKA tunnel shortest ipv4 from R5 -> R6
 tunnel vrf v1
 tunnel source loopback0
 tunnel destination 20.20.20.6
 tunnel domain-name 20.20.20.1 20.20.20.2
 tunnel mode polka
 vrf forwarding v1
 ipv4 address 30.30.30.1 255.255.255.252
 no shutdown
 no log-link-change
 exit
```

Figure 5. IPv4 Tunnel settings

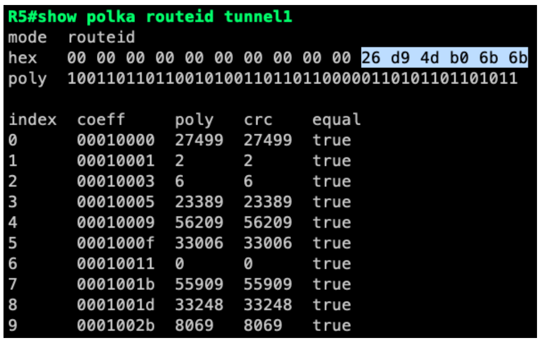


```
R5#show running-config interface tunnel4
interface tunnel4
 description POLKA tunnel longues ipv6 from R5 -> R6
 tunnel vrf v1
 tunnel source loopback0
 tunnel destination 2020::6
 tunnel domain-name 2020::1 2020::4 2020::3 2020::2
 tunnel mode polka
 vrf forwarding v1
 ipv6 address 4040::1 ffff:ffff:ffff:ffff::
 no shutdown
 no log-link-change
 exit
```

Figure 6. IPv6 Tunnel settings

Creating PolKA Tunnels: Two tunnels were created, one to cross the shortest path (2 hops) **Fig. 9** and a second one to cross the longest path (4 hops) with an example of settings in IPv6 **Fig. 10**. An important parameter is `tunnel domain-name`, that defines the path containing the address of all the routers through which the **PolKA**'s packets will cross. This command enables the Route-ID creation at the edge assigning it to the respective tunnel.

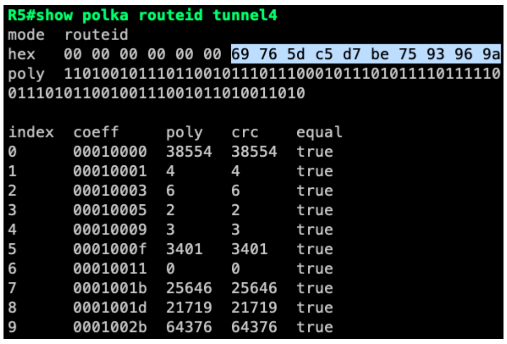
Visualizing tunnels in the edges: PolKA `routeID` is automatically computed when creating PolKA tunnels. **Fig. 7** and **Fig. 8** allow us to observe the configurations on the edge router R5. This tunnel inserts the `routeID` in the **PolKA**'s packet header. Then each router along the path extracts the `routeID` to forward the packet to the next hop, by using a modulo operation with `routeID` and its `nodeID`.



```
R5#show polka routeid tunnel1
mode  routeid
hex   00 00 00 00 00 00 00 00 00 00 26 d9 4d b0 6b 6b
poly  1001101101100101001101101100000110101101101011

index  coeff     poly   crc    equal
0      00010000  27499  27499  true
1      00010001  2      2      true
2      00010003  6      6      true
3      00010005  23389  23389  true
4      00010009  56209  56209  true
5      0001000f  33006  33006  true
6      00010011  0      0      true
7      0001001b  55909  55909  true
8      0001001d  33248  33248  true
9      0001002b  8069   8069   true
```

Figure 7. RouteID on Tunnel 1



```
R5#show polka routeid tunnel4
mode  routeid
hex   00 00 00 00 00 00 69 76 5d c5 d7 be 75 93 96 9a
poly  11010010111011001011101110001011101011110111110
01110101100100111001011010011010

index  coeff     poly   crc    equal
0      00010000  38554  38554  true
1      00010001  4      4      true
2      00010003  6      6      true
3      00010005  2      2      true
4      00010009  3      3      true
5      0001000f  3401   3401   true
6      00010011  0      0      true
7      0001001b  25646  25646  true
8      0001001d  21719  21719  true
9      0001002b  64376  64376  true
```

Figure 8. RouteID on Tunnel 4

Repository and dissector: A GitHub[5] was created to allow the reproducibility of **PolKA** experiments on **FreeRouter**. All the files and guidelines are available in this repository, ranging from installing **FreeRouter** to executing the topology described in the paper. In addition, there are commands that allow testing, viewing and debugging **PolKA**. Therefore, we show tunnel configuration and connectivity tests.

A Wireshark dissector was built to facilitate the analysis of the **PolKA** protocol header, which allows us to capture and debug the protocol traffic interactively. In this GitHub[6],

[5]`https://github.com/eversonscherrer/wpeif2022`
[6]`https://github.com/eversonscherrer/dissector-polka`

we provide a HOWTO that contains all the necessary information for using and testing the protocol.

4.2. Agile Path Reconfiguration with Policy Based Routing

In order to demonstrate an agile path reconfiguration, we make use of a policy based routing (PBR) for traffic classification at the edges. So, firstly, the traffic is classified through an access list, and then it is forwarded through one of the available tunnels. This allows us steering the traffic quickly through the **PolKA** tunnels to balance the traffic avoiding e.g. congestion due to an unequal use of the network.

The **Figs. 9 and 10** also includes the router R7 to generate traffic from R7 so that the router R5 applies a specific PBR (Policy-Based Routing) within **PolKA** protocol domain. It opens up opportunities to make traffic engineering (TE) for network performance improvement.

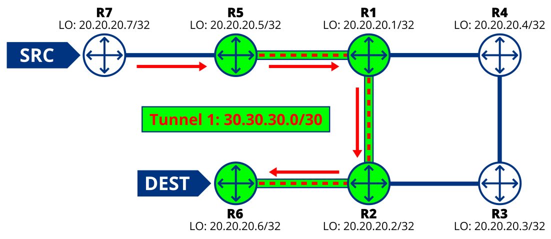


Figure 9. Shortest Path

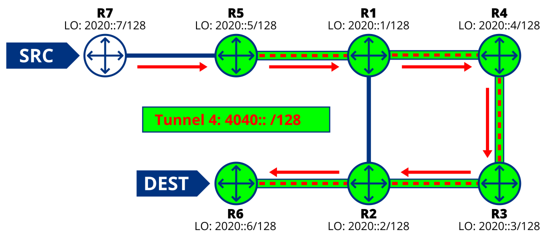


Figure 10. Longest Path

As mentioned before, the complete topology is available in GitHub with four **PolKA** tunnels. However, we demonstrate a shortest path with an ipv4 tunnel and a longest path with ipv6 tunnel. In **Fig. 9**, we show a traffic in green sent between edge router R5, passing through core routers R1 and R2, to reach at edge router R6 over **PolKA** tunnel 1. **Fig. 10** shows a **PolKA** tunnel in IPv6 to transport traffic over the longest path crossing the core routers R1, R4, R3, and R2 until reaching the edge router R6. Load balancing, traffic engineering are applications that may exploit **PolKA** properties offering PBR and routing control configurable at the source.

5. Conclusion

In this work, we describe the **PolKA** architecture and its protocol implementation within **FreeRouter** open-source router platform. We carried out experiments deploying **PolKA** in a topology composed by edge and core nodes. All the required topological information for **PolKA** was enabled by reusing the OSPF legacy protocol. For validation purposes, tunnels were set both with IPv4 and IPv6 to reach at distinct destinations through different paths demonstrating that connectivity service is functioning properly.

As lessons learnt, the lifecycle experience from prototyping to deployment is a loop-intensive development process. The right balance between implementing everything from scratch as opposed to reusing legacy protocols is key in this process. Although open-source router platforms such as **FreeRouter** bring along legacy protocols, it still needs to improve design modularity and a minimal set of documentation for developers willing to integrate and test their own protocol implementations.

References

Bajard, J. C. (2007). A residue approach of the finite fields arithmetics. In *2007 Conference Record of the Forty-First Asilomar Conference on Signals, Systems and Computers*, pages 358–362.

Dominicini, C. et al. (2020). Polka: Polynomial key-based architecture for source routing in network fabrics. In *2020 6th IEEE Conference on Network Softwarization (NetSoft)*, pages 326–334. IEEE.

Dominicini, C. et al. (2021). Deploying polka source routing in p4 switches. In *2021 International Conference on Optical Network Design and Modeling (ONDM)*, pages 1–3. IEEE.

Liberato, A. et al. (2018). RDNA: Residue-Defined Networking Architecture Enabling Ultra-Reliable Low-Latency Datacenters. *IEEE TNSM*.

Martinello, M. et al. (2014). KeyFlow: a prototype for evolving SDN toward core network fabrics. *IEEE Network*, 28(2):12–19.

Shoup, V. (2009). *A computational introduction to number theory and algebra*. Cambridge university press.