M-PolKA: Multipath Polynomial Key-based Source Routing for Reliable Communications

SUBMISSION DATE / POSTED DATE

02-02-2022 / 01-03-2022

CITATION

Silva Guimaraes, Rafael; Klippel Dominicini, Cristina; Manuel Garcia Martínez, Víctor; Missi Xavier, Bruno; Rossi Mafioletti, Diego; Locateli, Ana Claudia; et al. (2022): M-PolKA: Multipath Polynomial Key-based Source Routing for Reliable Communications. TechRxiv. Preprint. https://doi.org/10.36227/techrxiv.19072946.v2

DOI

10.36227/techrxiv.19072946.v2

M-PolKA: Multipath Polynomial Key-based Source Routing for Reliable Communications

Rafael S. Guimarães, Cristina Dominicini, Víctor M. G. Martínez, Bruno M. Xavier, Diego R. Mafioletti
Ana C. Locateli, Rodolfo Villaca, Magnos Martinello, and Moisés R. N. Ribeiro

***Abstract*—Innovative traffic engineering functions and services require disrupting routing and forwarding mechanisms to be performed with low overhead over complex network topologies. Source routing (SR) is a prominent alternative to table-based routing for providing the needed expressiveness and agility by reducing the number of network states. This work proposes the M-PolKA, a topology-agnostic multipath source routing scheme and orchestration architecture for reliable communications, which explores special properties from the Residue Number System (RNS) polynomial arithmetic. A P4-based proof-of-concept is experimentally demonstrated using emulated and hardware prototypes. Also, use cases for revealing M-PolKA's functionalities are tested in different scenarios in order to address problems, such as communication reliability improvement, agile path migration and fast failure reaction. Finally, low overhead for extra functionalities is observed when RNS-based SR is compared to traditional routing approaches.**

***Index Terms*—source routing, residue number system, SDN, Chinese remainder theorem, reliability, multipath.**

I. Introduction

In recent years, we have seen a rising interest in the evolution of a new generation of network architectures, thanks to the emergence of network programmability enabled by Software-Defined Networking (SDN) [1] and Network Function Virtualization (NFV). In this sense, SDN and NFV are enablers for the management and control of different technologies in order to meet stringent end-to-end (E2E) service requirements. Indeed, they are essential to tackle modern challenges in 5G networks, such as selecting routing paths in an agile and efficient manner for scenarios with high mobility [2], and adapting to variable traffic patterns [3].

Nowadays, networks are characterized by topologies with high path diversity, which can be exploited to route traffic across diverse paths. Multipath routing enables efficient usage of the available bandwidth offering high-quality network services, and may increase the reliability and resilience of communications [4]. The new generation of networks requires abilities for routing and forwarding packets under strict reliability requirements, which introduces the need and the complexity of managing multiple paths in the network.

In many cases, the multipath routing proposed in the literature is difficult to implement due to the diversity of technologies and processes involved [5]. However, the architectural disaggregation of the planes from SDN opens up new opportunities. On the one hand, the control plane is in charge of implementing more efficient algorithms to compute multiple paths for a flow. On the other hand, programmable data planes deal with packet forwarding and traffic splitting schemes across multiple paths.

A major challenge to support an efficient multipath routing is the inherent data and control planes scalability limitations in a legacy SDN table-based approach. From the data plane side, as the switching hardware supports limited table sizes, the growing number of table entries leads to scalability and performance concerns. From the control plane side, path diversity brings the challenge of dealing with a large number of control messages needed to reconfigure the network and consequently keep the multipath state. This is far from a trivial problem due to the scale, dynamics, heterogeneity, mobility, and high-performance required by modern applications, such as Ultra-reliable and Low Latency Communication (URLLC) [6] [7].

In this context, source routing (SR) schemes for multipath forwarding [8] arise as strong candidates to replace table-based routing [9], [10], since they avoid the reconfiguration of all the nodes along the path. These schemes allow traffic engineering to dynamically exploit all existing paths to achieve maximum throughput [11]. SR also reduces the control signaling and latency related to path setup convergence, so that migrating paths is only a matter of changing the state at source or edge nodes[1]. However, traditional multipath SR solutions lack expressiveness, depend on updating a list state in the packet, and do not offer intrinsic mechanisms to react to failures [12].

In this paper, we want to push to an extreme design choice, and answer the following questions: (i) is it possible to define a fully stateless multipath SR approach (i.e., no state in the core nodes, nor in the packet) in a network fabric that intrinsically enables reliable communication via exploitation of path diversity (i.e., does not depend on upper layers)?; and (ii) how to implement such approach in commodity network hardware with support to any topology?

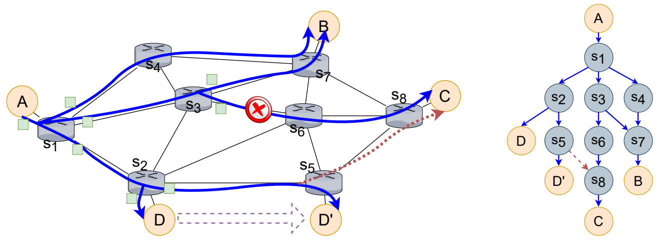
To this end, we propose a topology-agnostic multipath SR scheme and orchestration components for reliable communications, named **M-PolKA (Multipath Polynomial Key-based Architecture)**, which explores special properties from the Residue Number System (RNS) with polynomial arithmetic using Galois field (GF) of order 2 [13], known as GF(2).

Rafael S. Guimarães, Cristina Dominicini Bruno M. Xavier and Diego R. Mafioletti are with the Federal Institute of Education, Science and Technology of Espírito Santo Espírito Santo, Brazil (e-mail: rafaelg@ifes.edu.br).

Ana C. Locateli, Víctor M. G. Martínez, Rodolfo Villaca, Magnos Martinello, and Moisés R. N. Ribeiro are with the Federal University of Espírito Santo, Espírito Santo, Brazil.

Manuscript received month XX, 202X; revised month XX, 202X.

[1] The edge node may be a virtual switch in a server, a hypervisor, a top-of-rack (ToR) switch, or an ingress domain gateway.



(a) Scenario (b) Multipath tree

Figure 1: Multipath routing for reliable communications.

These properties can be applied to any network topology and guarantee that the node sequence is irrelevant to derive the route label, which remains unchanged throughout all the path [14], [15]. In this scheme, at core nodes, the transmission states of the output ports are given by the remainder of the binary polynomial division (i.e., a mod operation) of the route identifier of the packet by the node identifier.

M-PolKA is a generalization of our previous work, named PolKA [12]. It redefines the RNS coding representation for multipath trees, in contrast to the single path routing of PolKA. As a consequence, the result of the forwarding operation is not an output port, but it gives the transmission states of the ports. In this way, M-PolKA offers reliability features with the specification of a tree or even a specific branch that can be quickly configured by only setting a route label in the edge. Furthermore, a packet duplication mechanism has been added to the M-PolKA switches to support multipath routing.

In summary, the contributions of this work are: (i) we propose M-PolKA, a topology-agnostic RNS-based multipath SR scheme and an orchestration architecture for network fabrics that is compatible with binary polynomial arithmetic; (ii) we implement such scheme in P4-enabled programmable switches [1] by reusing the Cyclic Redundancy Check (CRC) hardware to enable the polynomial *mod* operation; (iii) we implement emulated and hardware prototypes to demonstrate that M-PolKA can achieve similar performance to traditional approaches; and (iv) using programmable networks as a proof-of-concept, we demonstrate how M-PolKA can explore path diversity and RNS properties to enable intrinsic reliability features in the network fabric, like packet duplication, agile path selection and fast failure reaction.

The remaining of this work is organized as follows. Section II discusses the problem statement and related works. Section IV-A presents the M-PolKA architecture. In Section III introduces the source routing scheme used in our proposal, and performs a scalability analysis. Section IV studies how to implement M-PolKA according to P4 data plane architecture, followed by the implementation and evaluation of proof-of-concept prototypes in Sections V and VI. Then, Section VII discusses conclusions and future works. Finally, Annex I presents the mathematical background of M-PolKA.

II. Problem Statement and Related Work

This section details the scope and contributions of our proposal considering the state-of-the-art.

A. Problem definition

In the context of computer networks, path diversification has been studied to solve several problems, such as load balancing, congestion control, reliability, Quality of Service (QoS), and security [8]. Our specific objective is to investigate how a multipath forwarding strategy in the lower layers can help network operators to achieve better reliability. To this end, the following requirements are important:

- **Expressiveness**: Is it possible to represent any set of paths to distribute packets to single or multiple endpoints?
- **Agility**: What is the convergence time to modify the paths, and apply these changes in all the affected nodes?
- **Low overhead**: What are the control plane and packet duplication overheads?
- **Topology-agnostic**: Is it possible to enable multipath routing in any topology?

Fig. 1(a) describes a high level scenario to show some functionalities that can improve the reliability of the system by exploring the multipath routing mechanism capacities, and Fig. 1(b) shows the correspondent multipath provisioning tree:

- **Seamless mobility between endpoints:** If node D migrates from S_2 to S_5 (e.g., handover scenarios), the content may be proactively (or reactively) delivered to both endpoints in order to enable reliable communications.
- **Packet duplication for failure protection**: By providing several (preferable disjoint) paths from source to destination and sending the same packet through each of them, the communication remains uninterrupted in case of failure in one of the paths. For example, node A sends duplicated packets to node B over two paths ($S_1 - S_3 - S_7$ and $S_1 - S_4 - S_7$). Thus, if one path fails, the packets will still reach the destination.
- **Agile path migration for failure reaction or QoS**: By adding the link $S_5 - S_8$ in the multipath provisioning tree, it is possible to enable a new path to node C ($S_1 - S_2 - S_5 - S_8$). If this change can be quickly configured, the network can react to a failure of the link $S_3 - S_6$ or a performance degradation of the path $S_1 - S_3 - S_6 - S_8$ (e.g., concurrent flows sharing some links).
- **Content distribution to multiple endpoints with minimal overhead:** Node A distributes a flow to nodes B, C and D over multiple paths with no need to duplicate packets in shared links (e.g., $S_1 - S_3$ and $S_1 - S_2$).

B. Multipath routing and scope definition

Fig. 2 presents a classification of multipath protocols and highlights the scope of this paper (in yellow). Several layered-based protocols have been proposed to explore the maximum benefit of multiple paths with least overhead, but each layer presents some advantages and drawbacks. In general, the higher layers (e.g., transport or application layers) have a poor knowledge of the underlay network, and depend on the lower layers to share information regarding the path diversity [8].

For example, in [16], authors discuss the advantages of Multipath TCP (MPTCP) aiming to boost application network performance by the aggregation of bandwidth over multiple network links, employing SDN to track the available

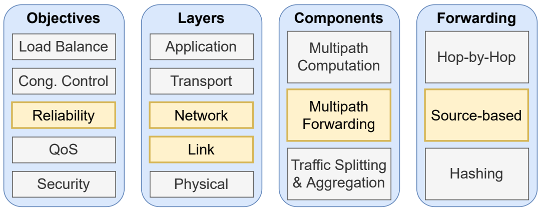


Figure 2: The scope of this work (in yellow), considering the classification of multipath protocols.

bandwidth of the connected links and selects the best path. However, as shown in [17], as the number of failed links increases, MPTCP can only recover full throughput if the link failure occurs on the server side.

On the other hand, traditional network and link protocols (e.g., OSPF or STP) suffer many limitations for supporting multipath provisioning [8]. For instance, Dinh et al. [18] discuss that in classical IP networks, IP packets are mostly forwarded using the shortest path toward the destination. However, in the scenario of network with several spare links, a link failure occurs, the router in the network would immediately calculate a new shortest path and shift the network data over to the new link. This would lead to the new path becoming congested, while the other spare links would not be utilized to lessen the congestion. Nonetheless, the total time to recalculate the new route can impact latency-sensitive network traffic.

As shown in Fig. 2, our proposal is a combination of layers 2 and 3, but it leverages its own source routing approach over programmable data planes. Thus, M-PolKA differs from most of the related works, because it does not depend on upper layers to enable multipath routing and reliability features, and is not built upon traditional network and link protocols.

Fig. 2 shows the three basic components of a multipath protocol [4], [8]: (i) *multipath computation* algorithms, with a global view of the network topology, compute multiple node/link-disjoint paths for a given traffic flow in order to improve fault-tolerance and offer more aggregate bandwidth; (ii) *multipath forwarding* algorithms forward packets on diverse paths, by mapping incoming packets to outgoing links; (iii) *traffic splitting and aggregation* algorithms split and combine traffic across multiple paths according to different algorithms (e.g., Round-Robin or Per-flow) [4]. M-PolKA focuses on the multipath forwarding mechanisms to forward packets along multiple paths. Although it is not in the scope of this work, it can exploit multipath computing algorithms to find node-disjoint paths as well as link-disjoint paths between the end-hosts or even traffic splitting strategies [8].

The last column of Fig. 2 presents some of the well-known multipath forwarding mechanisms: (i) *destination-based (hop-by-hop) forwarding*, as in today's IP networks; (ii) *source-based routing* approach, in which the responsibility of defining the route belongs to the source of packets, which can specify all the elements of the path to the destination; and (iii) *hashing*, in which the router performs a hash on the packet's header, and the outcome of the hash function gives the output port.

M-PolKA does not suffer from the limitations of per-hop table-based methods, such as the scalability of the number of states and latency for path modification. This is achieved by embedding the paths inside the packets in a arithmetic fashion that can be decoded by the programmable switch hardware.

C. Source routing schemes

1) Single path SR: Many works investigated the benefits of SR over traditional table-based routing, such as massive reduction of network states, and optimal use of network capacity [19] [11] [20]. The most traditional way of executing source routing is a list-based SR (LB-SR) approach, in which the route label represents a ordered list (or stack) of output ports, and the forwarding operation is a pop of the first element [19]. Although this approach drastically reduces the burden of managing network states by eliminating tables in core nodes, it still needs to maintain a state in the packet by using a route label rewrite operation in every node to update the position in the list. This operation may be costly for packet networks and difficult to implement in optical networks [15].

A classical problem in SR is how fast it reacts to node or link failures [21]. When a failure is detected, the source calculates a new route label, but this change takes a long time leading to a loss of packets in transit. A solution is to embed alternate paths in the nodes, but it increases the number of network states. An alternative is to embed paths in the packets, but the flat list encoding does not offer an implicit way of representing additional paths.

Recent works [22] [12] have brought RNS-based SR as an alternative method to perform SR that defines the outgoing port on each node by using modulo operation between the route label and a node identifier. Related works integrated RNS with SDN [14], developed fast failure reaction mechanisms [21], investigated techniques to improve the scalability of the *routeID* [23], and applied the scheme to enable SFC [24]. However, all these works rely on integer RNS arithmetic, and the integer mod operation cannot be implemented in current commodity network hardware. Therefore, they either use software switches implementations [14], [24], or depend on synthesizing integer division to ASICs or NetFPGAs [22].

PolKA [12] differs from previous RNS-based SR works by bringing a broader SR expressiveness that is closer to elementary binary polynomial operations. The immediate benefit of PolKA is to enable the reuse of commodity embedded network functions that are based on polynomial arithmetic. For instance, CRC hardware provides wire-speed implementations of a mod operation [25], and is evolving for supporting configurable polynomials [26], [27]. Another work of the authors [28] deployed PolKA SR in a continental testbed composed of P4-enabled Tofino switches, demonstrating PolKA matches the data plane performance of traditional approaches. PolKA also provides a fast failover mechanism, which explores deflection routing for specifying additional protection nodes that force the packets back to the original single path. However, with PolKA, there is no way to define a route label that represents a multipath tree for the main path.

2) Multipath SR: The single path SR approaches can be extended to enable multipath routing, if the list with port labels is replaced by the transmission state of the output ports of each node. In this way, each core node can extract an array

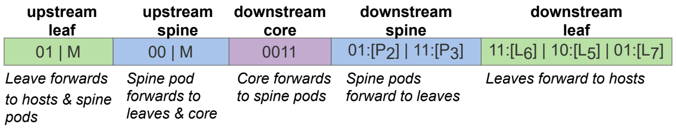
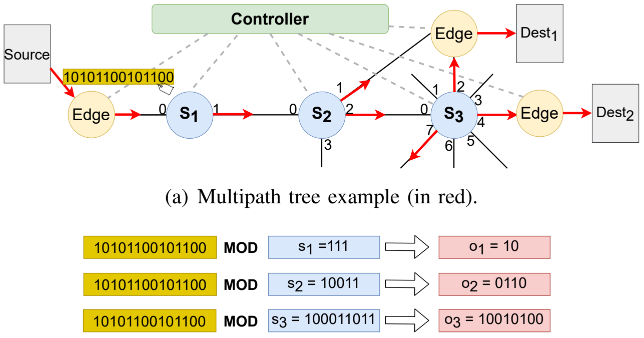

upstream leaf
upstream spine
downstream core
downstream spine
downstream leaf
01 | M
00 | M
0011
01:[P2] | 11:[P3]
11:[L6] | 10:[L5] | 01:[L7]
Leave forwards to hosts & spine pods
Spine pod forwards to leaves & core
Core forwards to spine pods
Spine pods forward to leaves
Leaves forward to hosts

Figure 3: Example of ELMO's header. Adapted from [9].

Table I: Comparison of multipath routing methods

Routing method	State at core	Forwarding operation	Comm. hard.?	Topo. Agnostic?
Table-based Multipath	table	lookup	✓	✓
List-based SR (LB-SR) [9]	packet	pop	✓	×
Hybrid Multipath SR [10]	packet table	pop lookup	✓	✓
Integer RNS-based SR [29] [22]	-	integer mod	×	✓
M-PolKA RNS-based SR	-	**polynomial mod**	✓	✓



(a) Multipath tree example (in red).

(b) M-PolKA multipath forwarding

Figure 4: M-PolKA operation.

representing which of its ports should duplicate the received packet and transmit it over multiple paths.

To the best of our knowledge, aside from RNS-based SR, there is no other multipath SR method in the literature that is topology-agnostic, while not depending on tables in the core network. This is justified by the fact that the encoding of a multipath route label depends on the flattening of a tree, which is a non-linear structure. Thus, flattening demands a mapping into a vector structure with explicit index calculations to track the parent child relation, or even recursive mechanisms [30]. This issue is particularly critical for networking applications that operate at line rate. To tackle this problem, the related works either propose solutions tightly coupled to specific topologies with predefined characteristics [9], or explore hybrid approaches that combine tables with SR [10].

In this direction, ELMO [9] enables multipath SR, but it is not topology-agnostic, since it is strictly designed for a three-tiered Clos topology. Fig. 3 shows an example of how complex and purpose-specific is its encoding of the multipath tree. The header is formed by upstream (leaf and spine) and downstream (core, spine, and leaf) packet rules (p-rules), which encode a set of output ports as a bitmap to provide the multipath forwarding in a given node. For example, in the first p-rule (upstream leaf), the leave connected to the source host transmit to the neighbor host (01) and to all the connected spine pods (M). Then, in the next p-rule (upstream spine), the spine pods don't transmit to any leaves (00), but transmit to all the core pods (M). Similarly, the remaining fields define the downstream transmission in each part of the tiered topology.

In contrast, BIER [10] proposed a hybrid multipath SR approach: each switch has a table to define the next-hop (configured using an IGP protocol), and the route label determines a list of switches that the packet must go through. Depending on the next hop, the data plane can modify the route label, and clone the packet to one or many interfaces. However, to modify paths, the control plane may need to configure the states in the tables along the path.

In this context, the exploitation of RNS properties appears as an innovative alternative to solve this encoding of a multicast tree in SR. Previous works explored integer RNS-based SR [29] [22] for providing multicast communication, but they cannot be deployed in commercial programmable switches and do not explore the role of multipath routing for reliability.

Table I summarizes a comparison of multipath routing methods, considering where the *forwarding state* is stored in the core nodes, what is the *forwarding operation*, whether this operation is implemented in *commodity network hardware*, and whether it is *topology-agnostic*. Note that only RNS-based methods offer topology-agnostic and fully stateless multipath SR. However, M-PolKA takes a step further by exploiting polynomial RNS-based arithmetic to represent a multipath SR scheme that can be deployed in commodity switches. In addition, M-PolKA provides an orchestration architecture and a data plane implementation that enables reliability features.

III. M-PolKA Forwarding

This section introduces the M-PolKA principles that explore polynomials over GF(2) and the RNS system.

A. Multipath source routing scheme

Annex I presents the mathematical background that supports our proposal. In this paper, all polynomials will be considered as polynomials over GF(2). Note that a polynomial $f(t) = a_n t^n + a_{n-1} t^{n-1} + \ldots + a_1 t^1 + a_0 t^0$ can be represented by the bit string $a_n a_{n-1} \ldots a_1 a_0$. Thus, an identifier is represented by a bit string formed by the coefficients of a polynomial, which are either 0 or 1, and the bit length of the identifier is $len(f)$.

As presented in Fig. 4(a), M-PolKA architecture is composed by core nodes (in blue), edge nodes (in yellow), and a logically centralized controller (in green). In this architecture, the multipath SR relies on three polynomial identifiers over GF(2): (i) *routeID*: a multipath route identifier, calculated by the controller using the polynomial Chinese Remainder Theorem (CRT) and embedded into the packet by the edge nodes; (ii) *nodeID*: an identifier previously assigned to core nodes by the controller in a network configuration phase; and (iii) *portID*: an identifier assigned to the state of the output ports of each core node.

Let S $= \{s_1(t), s_2(t), \ldots, s_N(t)\}$ be a set of polynomials representing the *nodeID*s. The set S must be composed of pairwise co-prime polynomials, so we assume that $s_i(t)$ are irreducible polynomials, and satisfy the condition $deg(s_i) \geq nports$, where $nports$ denotes the number of ports in the node.

Let O = $\{o_1(t), o_2(t), \ldots, o_N(t)\}$ be the set of N polynomials $o_i(t)$ representing the *transmission state* of the ports in the node i (i.e., 0 do not transmit, 1 transmit a packet copy). For instance, the polynomial $o_i(t) = a_2t^2 + a_1t + a_0$ maps the state $a_2a_1a_0$, which means that there are 3 ports in the node s_i, and each coefficient represents the state of one port. If the output port polynomial is $o_i(t) = t^2 + t = 110$ then $a_2 = 1$, $a_1 = 1$, and $a_0 = 0$. This means that port 2 is transmitting, port 1 is transmitting, and port 0 is not transmitting.

A multipath is defined as the set S and its correspondent set O. The Controller calculates the *routeID* of a multipath using the polynomial CRT (see Annex I) as the polynomial $R(t)$ that satisfies:

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

Then, the forwarding operation in each core node calculates the transmission state of the output ports as the remainder of the euclidean division of the *routeID* in the packet by its *nodeID*: $o_i(t) = < R(t) >_{s_i(t)}$

B. Operation

Fig. 4 shows an example of M-PolKA operation for a topology composed of three core nodes. In a network configuration phase, the Controller calculates and assigns the *nodeID*s. The degrees of the polynomials assigned to switches s_1, s_2, and s_3 must be equal or greater than 2, 4, and 8, respectively. Consider the following irreducible polynomials assigned to the s_i nodes:

$$s_1(t) = t^2 + t + 1 = 111$$
$$s_2(t) = t^4 + t + 1 = 10011$$
$$s_3(t) = t^8 + t^4 + t^3 + t + 1 = 100011011$$

In this example, packets should be routed via a multipath represented in red in Fig. 4(a). The Controller may proactively compute $R(t)$ or calculate it when the first packet of a flow arrives. Considering the nodes s_1, s_2, and s_3 and the transmission states of Fig. 4(a), the set O is composed by:

$$o_1(t) = t = 10$$
$$o_2(t) = t^2 + t = 0110$$
$$o_3(t) = t^7 + t^4 + t^2 = 10010100$$

Let $R(t)$ be the polynomial satisfying:

$$R(t) \equiv (t) \mod (t^2 + t + 1)$$
$$R(t) \equiv (t^2 + t) \mod (t^4 + t + 1)$$
$$R(t) \equiv (t^7 + t^4 + t^2) \mod (t^8 + t^4 + t^3 + t + 1)$$

As in CRT for polynomials (see Annex I), we have:

$$M(t) = (t^2 + t + 1) \cdot (t^4 + t + 1) \cdot (t^8 + t^4 + t^3 + t + 1)$$
$$m_1(t) = s_2(t) \cdot s_3(t) = (t^4 + t + 1) \cdot (t^8 + t^4 + t^3 + t + 1)$$
$$m_2(t) = s_1(t) \cdot s_3(t) = (t^2 + t + 1) \cdot (t^8 + t^4 + t^3 + t + 1)$$
$$m_3(t) = s_1(t) \cdot s_2(t) = (t^2 + t + 1) \cdot (t^4 + t + 1)$$

Algorithm 1 Computation of the maximum $len(R)$.

```
function MAXLEN($nports, diameter, size$)
    $mindeg \leftarrow nports$
    $nodelist \leftarrow generate\_irred\_poly\_list(mindeg, size)$
    $pathlist \leftarrow get\_last\_itens(nodelist, diameter)$
    $length \leftarrow 0$
    for $elem \in pathlist$ do
        $length \leftarrow length + deg(elem)$;
    return $length$                                  ▷ Maximum $len(R)$
```

And solving Eq. (6) of Annex 1, we find $n_i(t)$:

$$n_1(t) = t, \quad n_2(t) = t^3 + t + 1, \quad n_3(t) = t^6 + t^5 + t^4$$

Finally, we calculate $R(t)$ according to Eq. (3) of Annex 1:

$$R(t) = t^{13} + t^{11} + t^9 + t^8 + t^5 + t^3 + t^2 = 10101100101100$$

After the *routeID* computation, the Controller installs flow entries in the edge element to embed the *routeID* 10101100101100 in the packets of that flow. Therefore, each node can calculate its *portID* by dividing this *routeID* by its own *nodeID*, as shown in Fig. 4(b). For example, the remainder of $R(t)$ = 10101100101100 when divided by $s_2(t)$ = 10011 is $o_2(t)$ = 0110. Thus, in s_2, ports 1 and 2 transmit, while ports 0 and 3 do not transmit.

C. Scalability analysis of the routeID

The goal of this section is to investigate the overhead of our new scheme in the bit length of *routeID* in comparison to the Port Switching and integer RNS-based approaches. The bit length of $R(t)$, $len(R)$, in M-PolKA is given by the equation:

$$len(R) = len(< \tilde{R}(t) >_{M(t)}) \leq \sum_{i=1}^{N} deg(s_i) \tag{2}$$

Algorithm 1 shows a pseudo-code for computing the maximum $len(R)$, given: the number of ports in each node ($nports$), the number of nodes ($size$), and the topology diameter ($diameter$). For the sake of simplicity, we consider all nodes have the same number of ports. A list of *nodeID* polynomials ($nodelist$) is generated, which consists of $size$ irreducible polynomials with degree greater than or equal to the minimum degree ($mindeg$). Note that we select polynomials with the lowest possible degree (e.g., if $mindeg = 5$, we start assigning one of the 6 existing irreducible polynomials of degree 5 to nodes, and, if necessary, we use the 9 existing irreducible polynomials of degree 6, and so forth). Thus, $nodelist$ is already ordered by degree. Finally, we select the very worst case scenario in which the polynomials in $nodelist$ with the greatest degrees assigned to the nodes in the longest possible path (i.e., the diameter). To this end, we pick the x last elements of $nodelist$, where $x = diameter$, and calculate the maximum $len(R)$ according to Eq.(2).

Table II compares the scalability of M-PolKA and LB-SR for two common DC topologies (fat-tree and two-tier), and three continental backbone topologies [31] (ARPANET, and GEANT2). This topology set covers diverse properties for number of ports, diameter and size. For backbone topologies, as the number of ports varies per node, we considered $nports$ as the maximum number of ports of any node. The results for

Table II: Maximum $len(R)$ for example topologies.

Topology	$nports$	$diam.$	$size$	Bits for M-PolKA	Bits for LB-SR
Two-tier S16 L16*	24	3	32	72	72
Fat-tree 16 pods	16	5	320	80	80
ARPANET	4	7	20	44	28
GEANT2	8	7	30	56	56

* *Two-tier topology with 16 spine switches and 16 leaf switches.*

the integer RNS-based approach were omitted, because they are very close to M-PolKA. This means that using M-PolKA instead of the integer version does not incur in reserving extra bits for the *routeID* header.

The LB-SR method is a simplified implementation, where the array of output ports is replaced by an array of transmission states of each port for each node, considering all the nodes have the same number of ports. This LB-SR implementation consumes equal or less bits for representing the *routeID* than M-PolKA, but it does not solve the multipath SR problem for any topology. We chose this method for worst-case baseline comparison, as explained in more details in Section IV-C.

For the topologies under worst case analysis in Table 1, the maximum $len(R)$ results show that M-PolKA fits existing packet headers (e.g., 96 bits of Ethernet source and destination addresses, or a stack of MPLS labels with 20 bits per label). When deploying RNS-based SR, the cost to exploit RNS features may be to reserve more bits in the packet header for representing the *routeID*. Nevertheless, there are known techniques that make optimal assignment of *nodeID*s (avoiding the worst case scenario), and reduce *routeID* length [22], [23].

IV. M-PolKA Design and Implementation

This section studies how to design and implement M-PolKA architecture in a P4-enabled data plane [1].

A. M-PolKA Architecture

The M-PolKA architecture is composed of three separate layers as shown in Fig. 5:

1) Data plane layer: M-PolKA architecture assumes the existence of a physical topology with high path redundancy, and a data plane composed of P4-enabled programmable switches that perform the role of edge or core nodes.

2) Control and orchestration layer: It is composed of four modules that enable the multipath routing. The monitoring module collects the network topology and state, such as link failure and congestion, in order to provide a feedback loop for orchestration decisions. The path computation module is responsible for computing multiple paths for a flow, which may be node-disjoint, link-disjoint or non-disjoint. The node configuration module is responsible for setting the nodeIDs of the core nodes, and for inserting, deleting, and updating flow entries at the edge nodes. It abstracts all the procedures involved in the setup of those parameters independently of the specific interfaces of the data plane architectures. The polynomial calculation module calculates the *nodeID*s, *portID*s, and *routeID*s, according to the requests of the other modules.

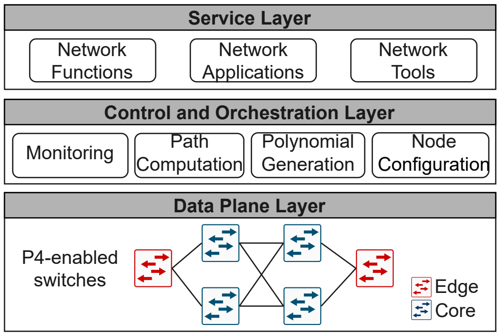


Figure 5: M-PolKA architecture.

3) Service layer: This layer is responsible for deploying network functions on top of the M-PolKA architecture. For example, functions to consolidate duplicated packets or re-ordering after multipath routing operations. Moreover, this layer is composed of all user applications and other tools used in the architecture implementation and validation stages.

B. M-PolKA data plane implementation

The M-PolKA header contains a *routeID*, whose maximum length depends on the network topology, as explained in Section III-C. Fig. 6(a) shows the format of M-PolKA header with a fixed length field for storing the *routeID*, after the Ethernet header. When the packet reaches an edge switch from an end-host, the `etherType` in the Ethernet header is TYPE_IPv4=`0x800`. Thus, the switch must parse the IPv4 header, encapsulate the SR header, and change the `etherType` to TYPE_SR=`0x1234`.

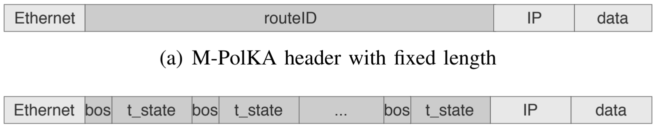
Each edge switch has a table, which is populated by the controller and maps destination IP address to their routing paths. The result of the table lookup is an action that sets the output port to the directly connected core switch and encapsulates a single *routeID*.

At core switches, the pipeline of Fig. 7(a) is executed. When `etherType` is TYPE_SR, as M-PolKA only needs read access to packet headers, it uses the `lookahead` method of P4 language that evaluates a set of bits from the input packet without advancing the packet index pointer. To discover the transmission state (`t_state`), the switch has to perform a mod operation between the *routeID* in the packet and its own *nodeID*. To iterate onto the transmission state, we use a chain of `if (<condition>)` statements for each port. We do not to use `resubmit` or `recirculate` actions, because we observed a huge overload in the latency when we tested them, since these built-in actions behave as repetition statements.

When the packet reaches an edge switch, the SR header is removed and the etherType is changed to TYPE_IPv4.

Fig. 7(b) shows the comparison with our previous PolKA work for single path routing, which uses the result of the mod operation for identifying the label of a single output port, and, therefore, cannot provide multipath routing.

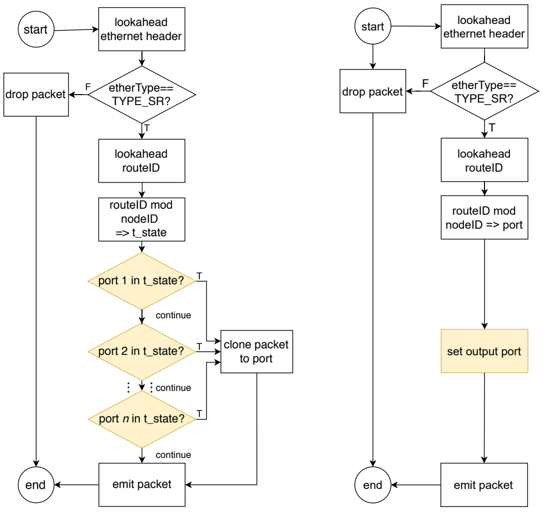
Polynomial or integer mod operations with non-constant operands are not natively supported by commodity network hardware and are not available in P4 language. To overcome this limitation, in [12], the authors developed a technique that



(a) M-PolKA header with fixed length

(b) Simplified LB-SR header with variable length

Figure 6: Source routing headers.



(a) M-PolKA

(b) PolKA

Figure 7: P4 core pipelines for M-PolKA and PolKA.

allows the execution of the polynomial mod in hardware by reusing common CRC operations, as follows:

1) $G = nodeID$, $r = degree(G)$
2) $D = routeID \div 2^r$ ***(SHIFT RIGHT)***
3) $dif = routeID - D * 2^r$ ***(SHIFT LEFT, XOR)***
4) $R =< D * 2^r >_G$ ***(CRC)***
5) $portID = dif + R$ ***(XOR)***

Therefore, the switch calculates the transmission state by using two *SHIFT*, one CRC, and two *XOR* operations. With the use of this technique, M-PolKA can be deployed in P4 targets that allow the configuration of generator polynomials. Since P4 supports CRC operations through the use of external libraries [1], the support for customized polynomials depends on the architecture models and how specific targets implement them. The `PSA`[2] and `v1model`[3] architectures support customized polynomials of 16 and 32 bits. In terms of targets, the software switch `bmv2`[4] and the hardware switch Tofino from Barefoot support customized CRC polynomials, while Netronome SmartNICs only support fixed CRC polynomials.

C. Comparison with list-based approaches

In contrast to M-PolKA, related works in the literature do not provide a multipath SR solution that is simultaneously topology-agnostic and tableless (see Section II-C2). A topology-agnostic multipath LB-SR solution involves mechanisms to flattening the tree structure and recursions, which would incur high performance costs. For the sake of comparison, we implemented a simplified version of a LB-SR solution, which uses a list of the transmission states of the ports of each node in the path. Although this simplified LB-SR solution does not solve the general multipath SR problem, it works for a linear branch use case, which is useful for the worst-case performance comparison presented in Section VI-B.

In our simplified LB-SR implementation, the edge nodes embed a list of SR headers to the original packet header, as shown in Fig. 6(b), and change the `etherType` to TYPE_SR=`0x1234`. Each item of the list has a `bos` (bottom of stack) bit and a transmission state (`t_state`). In the core switch pipeline, it parses the first SR header of the list, gets the `t_state`, and pops this header from the list. After acquiring the transmission state, the data plane clones the packet that is enqueued in the packet buffer to the correspondent egress ports. The `bos` is 1 only for the last hop, when the `etherType` is changed back to TYPE_IPv4.

It is important to note the following differences between the pipelines of M-PolKA and this simplified LB-SR solution:

- LB-SR header has variable size, while M-PolKA has a fixed-length header;
- In P4, LB-SR needs to create one encapsulating action for each list size (e.g., `add_header_1hop`, `add_header_2hops`, ...), which increases the number of code lines and memory for deploying the edge code;
- In LB-SR, each core switch rewrites the header to update the list, while M-PolKA does not change the *routeID*;
- In LB-SR, the transmission state is directly available in the header, while M-PolKA uses an arithmetic operation.

V. Proof-of-concept Prototypes

To evaluate the main functionalities of M-PolKA, two prototypes were implemented: (i) an emulated setup, and (ii) a physical setup that uses Netronome SmartNICs[5].

A. Emulated prototype

1) P4 architecture and target: The software switch `bmv2 simple_switch` with the `v1model` architecture was selected as the target for this prototype, once it supports all the functionalities required by M-PolKA, such as the configuration of CRC polynomials. Also, we use the `clone3` primitive action (P4_16) to enable the multipath feature. The number of clone session IDs is limited to 65535 in the `bmv2 simple_switch`. It is important to highlight that this software switch is a user-space implementation focusing on feature testing. There are other high performance implementations of P4 software switches and compilers (e.g., PISCES[6], P4ELTE[7], and MACSAD[8]), but they do not yet cover all the features required by M-PolKA. As these implementations arise, it will be possible to test our prototype with higher loads. For the time being, the solution was to limit the link rates to 10Mbps in our emulated prototype to avoid reaching the processing capacity limits of `bmv2 simple_switch`.

[2] https://p4.org/p4-spec/docs/PSA.html
[3] https://github.com/p4lang/p4c/blob/master/p4include/v1model.p4
[4] https://github.com/p4lang/behavioral-model
[5] https://www.netronome.com/
[6] http://pisces.cs.princeton.edu/
[7] http://p4.elte.hu/
[8] https://github.com/intrig-unicamp/macsad/

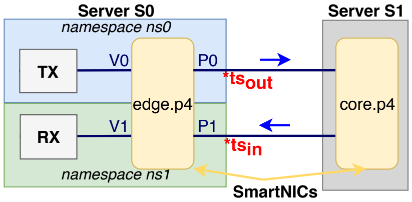


Figure 8: SmartNIC setup.

2) Setup description: The setup consists of one server Dell PowerEdge T430, with one Intel Xeon E5-2620 v3 2.40 GHz processor and 64 GB of RAM. To build our emulated environment, we used Mininet-WiFi [32] with P4, which becomes a tool that augments the Mininet emulator by including virtual wireless stations (STA) and access points (AP).

3) Control plane implementation: It has two main functionalities: (i) for the core: compute *nodeID*s, and configure core switches; and (ii) for the edge: compute routing paths for the traffic flows, calculate *routeID*s for these paths, and configure table entries in edge switches. For the RNS computation of *nodeID*s and *routeID*s, we developed a Python library[9] and application that uses the package `galoistools` of the `sympy` library[10] for GF(2) arithmetic operations. We also developed a control plane application in Python that communicates with the switches using the CLI provided by the `bmv2 simple_switch` and connects to a Thrift RPC server running in each switching process.

4) Header size: The `bmv2 simple_switch` supports the specification of CRC polynomials of 16 and 32 bits. Therefore, although our tests could use much smaller degrees, our PoC must adopt polynomials of degree 16. As the diameters of our test topologies are smaller than 10, the size of the M-PolKA header was defined as 160 bits. To a fairness comparison, we defined the LB-SR header as 16 bits (bos and port); therefore, for 10 hops, the array of headers has 160 bits.

B. Hardware prototype with SmartNICs

In the emulated prototype, the CRC operation is executed in software using CRC tables. However, the main benefit of using CRC is the execution of the mod operation in hardware with better performance. To this end, we built a hardware prototype.

1) Setup description: The setup is illustrated by Fig. 8 and consists of two servers: (i) S1: a device under test (DUT), running the core functionalities; and (ii) S0: a traffic generator (TG), running the edge functionalities, and transmitter (TX) and receiver (RX) functionalities in separate network namespaces. Both servers are Dell PowerEdge T430, with one Intel Xeon E5-2620 v3 2.40 GHz processor, 16 GB of RAM, and one Netronome Agilio CX 2x10GbE SmartNIC.

2) SmartNICs features: Netronome SmartNICs give access to hardware timestamps into the P4 programs.They partially implement the functionalities of *v1model* for P4 16, but it currently supports only a small set of fixed CRC polynomials, which restricts the use of its CRC hardware in M-PolKA. Nevertheless, we could use them for measuring forwarding latency in one core node, as proposed in the next section.

3) P4 programs: The P4 programs are adaptations of the codes used in the emulated prototype. The new edge code encapsulates both SR headers and a timestamp header for executing latency measurements, which is composed of two timestamps: ts_{out} and ts_{in} (see Fig. 8).

At server S0, packets are generated at TX and forwarded to the SmartNIC, where the edge adds SR (M-PolKA or LB-SR) and timestamp headers. When the packet leaves the edge, the hardware timestamp is assigned to ts_{out}. Then, it is transmitted to server S1 and processed by the core code at the SmartNIC, which is responsible for parsing the SR header and computing the output port. Since it is impossible to configure customized CRC polynomials in the SmartNICs, we use a standard CRC operation with a fixed CRC-16 polynomial. In this way, we execute all the M-PolKA pipeline steps (including the CRC operation) to measure their contribution to the total latency, As a result, the packet is delivered back to server S0, where the edge code assigns the value of the hardware timestamp to ts_{in}, removes the SR header, and delivers the packet to RX. Finally, all packets are captured with `tcpdump` tool, and parsed offline to extract the core forwarding latency for each packet using $ts_{in} - ts_{out}$.

VI. Evaluation

The first part of this section explores a software defined wireless network as a use case to demonstrate how M-PolKA can exploit path diversity for enabling reliable communication. To this end, three experiments showcase the following functionalities in our emulated prototype: seamless mobility, packet duplication for failure protection, and agile path migration for failure reaction and QoS. Then, we evaluate M-PolKA in comparison with the simplified version of LB-SR, (see Section IV-C) in the emulated and hardware prototypes.

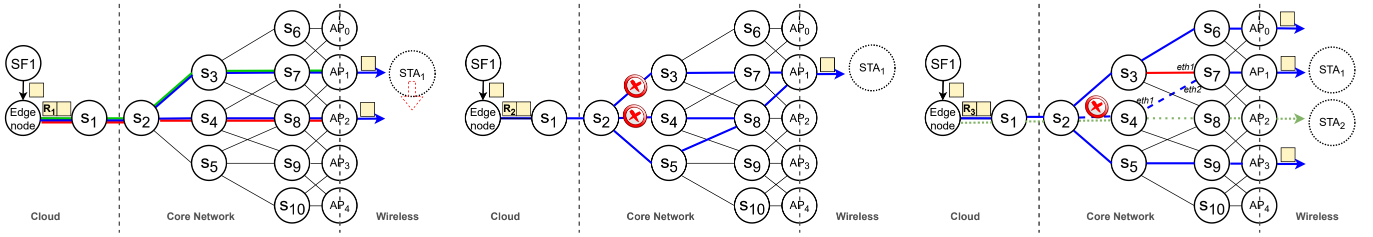
A. Emulated Prototype: Path diversity for reliability

Figure 9 shows the experiment scenarios with three network domains: cloud, core, and wireless. The cloud composes a virtualized environment responsible for providing service functions (SFs) to a specific mobile device group. In the core network, there are 9 switches (S$_n$), running core functionalities and five access points (APs) (AP$_i$) running edge functionalities. The wireless environment includes mobile elements (station - STA$_i$), where each STA has two wireless interfaces in order to provide multi-connectivity from different APs.

1) Seamless mobility: Figure 9(a) shows an example scenario to provide reliable communication between SF$_1$ and STA$_1$, when STA$_1$ migrates from AP$_1$ to AP$_2$. The list of switches for the segment (SF1 → STA$_1$) that is used for providing the multipath scheme is: S = $\{S_1, S_2, S_3, S_4, S_7, S_8\}$.

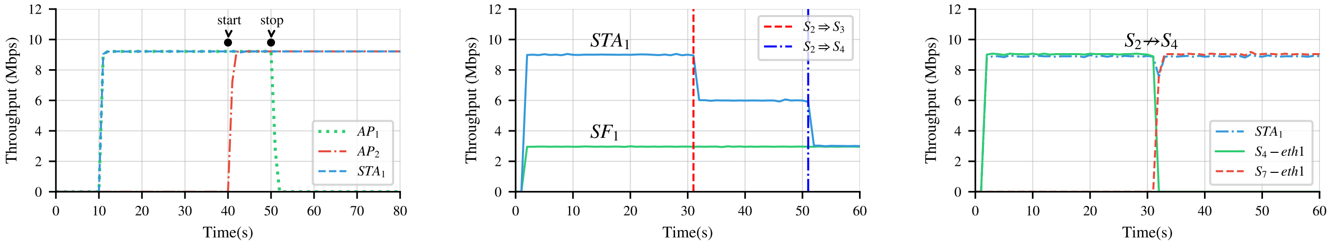
In this experiment, we deployed a VNF-based solution at the STA$_1$ as proposed by [7], which performs flow deduplication before delivering packets to the application layer. To detect the handover, we developed a wireless orchestrator that monitors the signal-to-noise ratio of the STAs and informs the Controller, which modifies a single flow entry at edge node to configure the *routeID*s accordingly. A *routeID* carries the set of core nodes and the transmission states of each node to define a specific multipath.

[9] https://pypi.org/project/polka-routing/
[10] https://www.sympy.org/



(a) Seamless mobility (b) Packet duplication for failure protection (c) Agile path migration for failure reaction and QoS

Figure 9: Path diversity experiment scenarios.



(a) Seamless mobility (b) Packet duplication for failure protection (c) Agile path migration for failure reaction

Figure 10: Path diversity experiment results: Throughput at the destination (STA$_1$).

The experiment involves modifying the *routeID* three times while STA$_1$ moves: (i) STA$_1$ is connected to AP$_1$ and the control plane sets the *routeID* to the green path ($\{S_1, S_2, S_3, S_7\}$), (ii) STA$_1$ starts to move towards AP$_2$ and, during the handover process, the control plane sets the *routeID* to the blue multipath ($\{S_1, S_2, S_3, S_7\}$ and $\{S_1, S_2, S_4, S_8\}$); and (iii) when the link becomes stable between STA$_1$ and AP$_2$, the control plane sets the *routeID* to the red path ($\{S_1, S_2, S_4, S_8\}$)).

Figure 10(a) shows throughput measurements using the `bwm-ng` tool at STA$_1$, AP$_1$, AP$_2$. At 10 s, we start a 10 Mbps UDP traffic from SF$_1$ to STA$_1$ passing through AP$_1$. At 40 s, the transmission state on node S$_2$ is initated to pass through $S_3 - S_7$–AP$_1$ and $S_4 - S_8$–AP$_2$. At STA$_1$, the traffic constantly perceives about 10 Mbps as the VNF discards duplicated packets before sending to the application.

Although there is a high probability of losses during the handover processes (i.e., the mobility interruption time), there was no perceived throughput degradation during the migration between APs. Therefore, our scheme could increase reliability of the system, since M-PolKA allows to dynamically change the paths and deliver the packets to both endpoints during the handover. Besides, there is no need to maintain the state in the whole path, because all the transmission states are included in the *routeID* and changing the *routeID* is just a matter of modifying a single flow entry in the edge node.

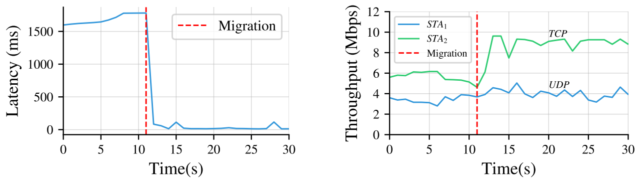
2) Packet duplication for failure protection: This experiment aims at demonstrating how the packet duplication feature can increase the reliability by exploring the path diversity, especially in the condition of multiple failures. As shown in Figure 9(b), the traffic between SF$_1$ and STA$_1$ can be steered via S$_3$, S$_4$ and S$_5$. Thus, we steer one packet to the path $\{S_1, S_2, S_5, S_8\}$ and two cloned packets to the paths $\{S_1, S_2, S_4, S_8\}$ and $\{S_1, S_2, S_3, S_7\}$.

Figure 10(b) shows the throughput measurements using the `bwm-ng` tool to expose the traffic perceived at STA$_1$. At 0s, SF$_1$ sends a 3Mbps UDP traffic, using the `iperf` tool, and the throughput at STA$_1$ is approximately 9 Mbps. Indeed, it is because of the traffic duplication across the three paths. At 31s, there is a failure between S$_2$ and S$_3$. Therefore, we observe a throughput of approximately 6 Mbps at STA$_1$. Then, at 51s, there is another failure between S$_2$ and S$_4$, and the throughput perceived at STA$_1$ is approximately 3 Mbps.

M-PolKA is capable of exploring path diversity by expressing in a single routeID a set of paths to steer the traffic. In this way, we can increase the communication reliability and the protection in case of successive events of failure.

3) Agile path migration for failure reaction and QoS: This experiment shows how the traffic can be steered using multipath SR with an agile path migration feature in the scenario of Figure 9(c). This feature can be used for improving reliability when the orchestrator identifies that a path migration can bypass failures or provide better performance.

In the first part of this experiment, SF$_1$ distributes a UDP traffic, using the `iperf` tool, to AP$_0$, to AP$_1$, and to AP$_3$. To this end, the Controller calculates a *routeID* for the multipath tree showed in blue in Figure 9(c), which comprises the nodes S = $\{S_1, S_2, S_3, S_4, S_6, S_7, S_9\}$ and their respective transmitting ports. Initially, the flow from SF$_1$ →STA$_1$ is allocated to the path (S_1–S_2–S_4–S_7–AP_1). Figure 10(c) shows the throughput measurements at STA$_1$, using the `bwm-ng` tool. At 30s, the Controller detects a link failure at link $S_2 - S_4$, and triggers a path migration to replace the branch $S_2 - S_4 - S_7 - AP_1$ with the branch $S_2 - S_3 - S_7 - AP_1$. To perform path migration, the Controller only has to modify a single flow entry at the edge node to set a *routeID* to steer the traffic through the red path. Then, once this single operation is executed, all the packets of the flow that leaves SF$_1$ are tagged with the *routeID* of the new path, and a small packet loss is detected at the destination.



(a) Latency at STA_2.

(b) Throughput at STA_2.

Figure 11: Agile path migration for QoS.

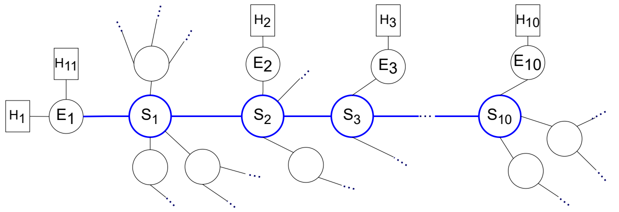


Figure 12: Analysis of the impact of the path length.

In the second part of this experiment, we show two examples of how M-PolKA can avoid latency and throughput degradation, by providing agile path reconfiguration. To demonstrate this feature, we include a concurrent traffic in green (from SF_1 to STA_2) in the scenario of Fig. 9(c). Consider the link between S_1 and S_2 has a bandwidth of 100 Mbps, while the links in the core network have a bandwidth of 10 Mbps.

Fig. 11(a) shows the results of a latency experiment, using the ping tool. We concurrently start a 10 Mbps UDP traffic in the blue path (SF_1 →STA_1) and a ICMP traffic in the green path (SF_1 →STA_2), as shown in Fig. 9(c). At 10 s, there is a huge latency between SF_1 and STA_2 (about 1600ms). In this experiment, the bottleneck is caused by the traffic concurrency in the link between S_2 and S_4 (10 Mbps). At 11 s, the control plane sets a different *routeID* for the UDP traffic, which implies to steer the UDP traffic to the red path (S_1-S_2-S_3-S_7). The ICMP traffic keeps using the same path (S_1-S_2-S_4-S_8). After migrating the UDP traffic, the Fig. 11(a) shows an abrupt drop to 15 ms in the latency at STA_2. Therefore, our scheme could fast react to performance degradation events by only modifying the *routeID* for a specific flow.

The last experiment explores the multiple links available in a given E2E communication to optimize the throughput between SF_1 and STA_2, as shown in Fig. 9(c). The measurements are carried out by using the iperf tool at SF_1. At 0s, as shown in Fig. 11(b), we start a 4Mbps UDP traffic (blue multipath: SF_1-S1-S_2-S_4-S_7-AP_1) and a TCP traffic to STA_2 (green path: SF_1-S1-S_2-S_4-S_8-AP_2). From 0s to 11s, the TCP traffic is competing with the 4 Mbps UDP traffic, as both flows are passing through the link S_2-S_4. Thus, the maximum throughput achieved by the TCP traffic is about 6 Mbps, limited by the 10 Mbps link limit between S_2 and S_4. At 11s, the controller decides to migrate the UDP traffic to a different path (SF_1-S1-S_2-S_3-S_7-STA_1), as shown in Fig. 11(b). After this migration, when the controller applies a different *routeID*, the experiment shows the throughput of the TCP traffic increases to more than 9 Mbps. With the multipath expressiveness of M-PolKA and the agile migration of SR, the traffic engineering can define diverse policies, such as link workload and flow priorities.

B. Comparison of M-PolKA vs. simplified LB-SR

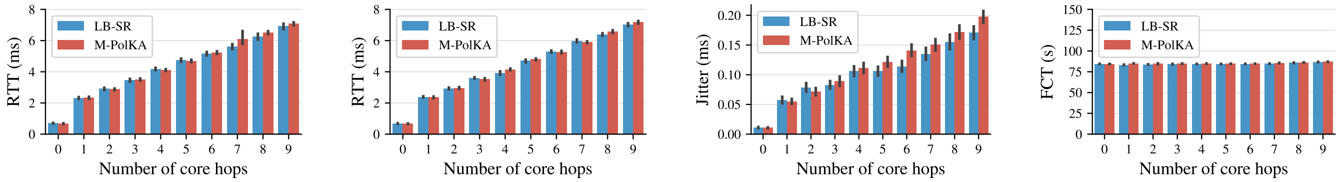
These experiments evaluate M-PolKA and the simplified LB-SR (see Section IV-C) for: (i) E2E tests in the emulated prototype, and (ii) a single-hop latency test in the hardware prototype. We considered Ethernet frames of 98 Bytes as small packets and frames of 1242 Bytes as big packets. The average and standard deviation are presented in all results.

1) E2E tests in the emulated prototype: The test uses the fabric topology of Fig. 12 to compare M-PolKA and LB-SR as the number of hops increases in the core network (e.g., from 0 for path $H_1 \rightarrow H_{11}$ to 9 for path $H_1 \rightarrow H_{10}$). The following experiments were executed: (i) round trip time (RTT): host H_1 sends 1 ICMP packet/s during 60 s to each of the other hosts using ping tool; (ii) jitter: host H_1 sends a UDP traffic of 5 Mbps (half of the link capacity) with big packets to each of the other hosts during 60 s using the iperf tool; and (iii) flow completion time (FCT): host H_1 transmits a file of 100 Mb with big packets over a TCP connection to each of the other hosts using the iperf tool (3 repetitions). Fig. 13 shows the comparison between M-PolKA and LB-SR solutions.

In the RTT experiment results, shown in the Fig. 13(a) and Fig. 13(b), it is possible to observe that the RTT grows linearly with the increase of the number of hops for both solutions. The LB-SR solution presents better RTT performance than M-PolKA solution, and for both solutions the standard deviation is small and in the same order of magnitude. Besides, there is no significant difference in the results for different packet sizes in the RTT experiments. This is because Mininet-WiFi+P4 does not consider the transmission time in the emulation. In addition, the jitter (Fig. 13(c)) is small and equivalent for both solutions. Finally, Fig. 13(d) shows that both solutions require approximately the same time to transfer the file and the standard deviation is small.

As discussed in Section IV-C, a topology-agnostic multipath LB-SR solution would be worse in terms of performance than this simplified LB-SR, which only works in a linear branch. Thus, we use this simplified version as a worst-case baseline for comparing the performance of M-PolKA, which offers a multipath SR solution that works for any topology. Therefore, we demonstrated that the *mod* operation over a fixed header can be executed with similar performance to the pop operation of list-based approaches. Nevertheless, the small difference between the two solutions can decrease if the CRC operation is performed in hardware, as shown in the next test.

2) Core latency tests in the hardware prototype: The goal of this experiment is to measure the core forwarding latency in M-PolKA and the simplified LB-SR when the path length increases. Since it is not possible to configure customized CRC polynomials in the SmartNICs (i.e., we can only use one *nodeID*), in each run, we execute a single hop forwarding in the core node, and vary the number of nodes included in the SR header to represent the path length. For each test execution, the traffic generator tool at TX varies the IP destination address. The last digit of the IP destination address represents the number of core nodes (e.g., if the IP destination is 10.0.100.1, the number of hops to the destination is 1, while IP destination 10.0.100.5 represents 5 hops to the destination).



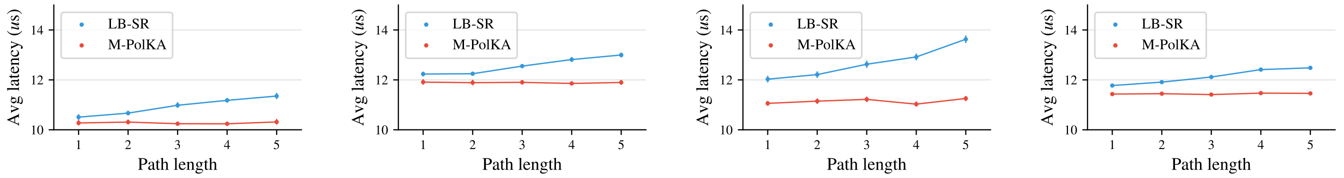
(a) RTT small packet

(b) RTT big packet

(c) Jitter

(d) FCT

Figure 13: Emulated prototype: Performance comparison between LB-SR and M-PolKA when the path length increases.



(a) Low throughput, small packets

(b) Low throughput, big packets

(c) High throughput, small packets

(d) High throughput, big packets

Figure 14: Hardware prototype: Comparison of the average core latency when the path length increases.

Depending on the number of hops, the edge encapsulates the appropriate SR headers (e.g., 5 hops, 5 SR headers in LB-SR).

The following experiments were executed for small and big packets: (i) low throughput: one ICMP pps, 100 packets in total, generated with `ping` tool; and (ii) high throughput: 1Gbps UDP packets, 1000 packets in total, generated with `pktgen` tool. Figure 14 compares the test cases for LB-SR and M-PolKA. In each test, the average latency and standard deviation in M-PolKA are small when the path length increases, while in an LB-SR, the average latency grows linear when the path length increases. This linear increase in latency measurements for an LB-SR is emphasized in the test case with high pps values (Fig. 14(c)) when the standard deviation is a bit higher for both M-PolKA and LB-SR due to the stress in edge and core elements. More investigation needs to be carried out in a hardware prototype that allows multi-hop tests, such as switches with Intel Tofino Architecture[11]. Despite this, the results collected so far indicate that M-PolKA implementation using CRC hardware is promising and can offer equivalent RTT and jitter performance to LB-SR.

VII. Conclusion

Herein, a binary polynomial representation of a fully stateless multipath SR approach, called M-PolKA, was proposed, implemented, and evaluated. The proposed architecture provides intrinsic reliability functionalities in the network fabric for any topology by exploring multipath routing and RNS properties. Moreover, our P4-based emulated and hardware prototypes demonstrated that is feasible to deploy RNS-based SR in commodity network equipment by reusing CRC hardware, with performance equivalent to traditional routing approaches. Reliability improvements and fast failover were validated using M-PolKA in a software-defined wireless networking scenario, demonstrating the potential to enable a new range of complex network applications.

Future works include the development of the algorithms for path computation and traffic splitting. Extensions of our hardware prototype for several scenarios using the Intel Tofino Architecture switches will also be evaluated. Moreover, the polynomial arithmetic of M-PolKA in hardware description languages has the potential to synthesize RNS-based SR in smaller chip areas and reduced clock cycles. Finally, we plan to extend our polynomial scheme using GFs of higher orders.

Annex I: Mathematical Background

This section describes the mathematical formulation that supports the proposal of M-PolKA. More information about finite fields and polynomials rings can be found in [33] [13].

Polynomial Ring over GF(2): Let GF(2) = $\{0, 1\}$ be the Galois Field of order 2, whose elements are residue classes modulo 2. The arithmetic operations of addition and multiplication in GF(2) are defined modulo 2. The set of all polynomials in one variable t with coefficients in GF(2), called polynomials over GF(2), is a ring considering the arithmetic operations of addition and multiplication modulo 2. If $f(t) = a_n t^n + a_{n-1} t^{n-1} + \ldots + a_1 t + a_0$ is a polynomial over GF(2), where $a_n \neq 0$, n is defined as the degree of $f(t)$, denoted by $deg(f)$. The length of $f(t)$, denoted by $len(f)$, is defined by $len(f) = deg(f) + 1$.

Euclidean Division Theorem for Polynomials: Let $f(t)$ and $g(t)$ be polynomials over GF(2) , where $g(t) \neq 0$. There exist unique polynomials $q(t)$ and $r(t)$ over GF(2) such that $f(t) = g(t).q(t) + r(t)$, where either $r(t) = 0$ or $deg(r) < deg(g)$. The polynomial $r(t)$ is called the remainder of the division of $f(t)$ by $g(t)$, and will be denoted by $< f(t) >_{g(t)}$.

Polynomial congruence: Given $f(t)$, $g(t)$, and $h(t)$ polynomials over GF(2), we say that $f(t)$ is congruent to $h(t)$ modulo $g(t)$, and write $f(t) \equiv h(t) \ mod \ g(t)$, if $h(t) =< f(t) >_{g(t)}$.

Irreducible Polynomials: A non-zero polynomial $g(t)$, is called a divisor of $f(t)$ over GF(2) if $f(t) = a(t).g(t)$, for some polynomial $a(t)$ over GF(2). Two polynomials $f(t)$ and $g(t)$ over GF(2) are coprime if their only common divisor is 1. A non-constant polynomial $f(t)$ over GF(2) is called irreducible over GF(2) if its only divisors are possibly a constant polynomial and itself.

[11] https://www.intel.com/content/www/us/en/products/network-io/programmable-ethernet-switch/tofino-series.html

Chinese Remainder Theorem (CRT) for polynomials: Let $s_1(t), s_2(t), \ldots, s_N(t)$ be monic pairwise coprime polynomials over GF(2) and let $M(t) = \prod_{i=1}^{N} s_i(t)$. There exists a unique polynomial $R(t)$ over GF(2) with $deg(R) < deg(M)$, satisfying $R(t) \equiv o_i(t)\ mod\ s_i(t)$, for $i = 1, 2, ..., N$, where:

$$\begin{aligned} R(t) &= < \tilde{R}(t) >_{M(t)} & (3)\\ \tilde{R}(t) &= \sum_{i=1}^{N} o_i(t) \cdot m_i(t) \cdot n_i(t) & (4)\\ m_i(t) &= M(t)/s_i(t) & (5)\\ n_i(t) \cdot m_i(t) &\equiv 1 \quad mod \quad s_i(t) & (6) \end{aligned}$$

The computation of $n_i(t)$ can be implemented using the Extended Euclidean Algorithm, which basically consists in applying the Euclidean Division Theorem several times. The algorithm complexity for computing $R(t)$ is $O\big(len(M)^2\big)$ [13].

Acknowledgment

This study was a recipient of the 2021 Google Research Scholar Award, and received funds from CAPES (Finance Code 001), CNPq, FAPESP, FAPES, CTIC, and RNP.

References

[1] The P4 Language Consortium, "P4 16 Language Specification." [Online]. Available: https://p4.org/p4-spec/docs/P4-16-v1.0.0-spec.html

[2] F. Z. Yousaf *et al.*, "NFV and SDN-Key technology enablers for 5G networks," *IEEE Journal on Selected Areas in Communications*, vol. 35, no. 11, pp. 2468–2478, nov 2017.

[3] I. F. Akyildiz *et al.*, "SoftAir: A software defined networking architecture for 5G wireless systems," *Computer Networks*, vol. 85, pp. 1–18, 2015.

[4] M. Suer *et al.*, "Multi-connectivity as an enabler for reliable low latency communications—an overview," *IEEE Communications Surveys Tutorials*, vol. 22, no. 1, pp. 156–169, 2020.

[5] Q. Wang *et al.*, "Implementation of multipath network virtualization with sdn and nfv," *IEEE Access*, vol. 6, pp. 32 460–32 470, 2018.

[6] P. Popovski, Č. *et al.*, "Wireless access in ultra-reliable low-latency communication (urllc)," *IEEE Transactions on Communications*, vol. 67, no. 8, pp. 5783–5801, 2019.

[7] R. S. Guimaraes *et al.*, "An sdn-nfv orchestration for reliable and low latency mobility in off-the-shelf wifi," in *ICC 2020 - 2020 IEEE International Conference on Communications (ICC)*, 2020, pp. 1–6.

[8] S. K. Singh *et al.*, "A survey on internet multipath routing and provisioning," *IEEE Communications Surveys Tutorials*, vol. 17, no. 4, pp. 2157–2175, 2015.

[9] M. Shahbaz *et al.*, "Elmo: Source routed multicast for public clouds," *IEEE/ACM Transactions on Networking*, pp. 1–14, 2020.

[10] I. Wijnands, E. C. Rosen, A. Dolganow, T. Przygienda, and S. Aldrin, "Multicast Using Bit Index Explicit Replication (BIER)," RFC 8279, "nov" 2017. [Online]. Available: https://rfc-editor.org/rfc/rfc8279.txt

[11] S. A. Jyothi *et al.*, "Towards a flexible data center fabric with source routing," in *Proceedings of the ACM SIGCOMM Symposium on SDN Research*. New York, NY, USA: ACM, 2015, pp. 10:1–10:8.

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

[13] V. Shoup, *A computational introduction to number theory and algebra*. Cambridge university press, 2009.

[14] M. Martinello *et al.*, "KeyFlow: a prototype for evolving SDN toward core network fabrics," *IEEE Network*, vol. 28, no. 2, pp. 12–19, 2014.

[15] H. Wessing *et al.*, "Novel scheme for packet forwarding without header modifications in optical networks," *Journal of Lightwave Technology*, vol. 20, no. 8, pp. 1277–1283, Aug 2002.

[16] H. Nam *et al.*, "Towards dynamic mptcp path control using sdn," in *2016 IEEE NetSoft Conference and Workshops (NetSoft)*, 2016, pp. 286–294.

[17] M. J.F. Alenazi, "Evaluating multipath tcp resilience against link failures," *The ISC International Journal of Information Security*, vol. 11, no. 3, pp. 113–122, 2019. [Online]. Available: http://www.isecure-journal.com/article_90849.html

[18] K. T. Dinh *et al.*, "Msdn-te: Multipath based traffic engineering for sdn," in *Intelligent Information and Database Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2016, pp. 630–639.

[19] X. Jin *et al.*, "Your data center switch is trying too hard," in *Proceedings of the ACM SIGCOMM Symposium on SDN Research*. New York, NY, USA: ACM, 2016, pp. 12:1–12:6.

[20] M. Soliman *et al.*, "Source routed forwarding with software defined control, considerations and implications," in *ACM Conference on CoNEXT Student Workshop*. New York, NY, USA: ACM, 2012, pp. 43–44.

[21] R. R. Gomes *et al.*, "KAR: Key-for-any-route, a resilient routing system," in *2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshop*, June 2016, pp. 120–127.

[22] A. Liberato *et al.*, "RDNA: Residue-Defined Networking Architecture Enabling Ultra-Reliable Low-Latency Datacenters," *IEEE TNSM*, 2018.

[23] Y. Ren *et al.*, "Flowtable-free routing for data center networks: A software-defined approach," in *GLOBECOM 2017 - 2017 IEEE Global Communications Conference*, Dec 2017, pp. 1–6.

[24] 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.

[25] W. W. Peterson *et al.*, "Cyclic codes for error detection," *Proceedings of the IRE*, vol. 49, no. 1, pp. 228–235, Jan 1961.

[26] M. Grymel *et al.*, "A novel programmable parallel crc circuit," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 19, no. 10, pp. 1898–1902, Oct 2011.

[27] Microchip, "32-bit programmable cyclic redundancy check (crc)," http://ww1.microchip.com/downloads/en/DeviceDoc/dsPIC33_PIC24-FRM,-32-Bit-Programmable-Cyclic-Redundancy-Check-(CRC)-DS30009729C.pdf, 2018, Accessed: 2019-01-13.

[28] C. Dominicini *et al.*, "Deploying polka source routing in p4 switches : (invited paper)," in *2021 International Conference on Optical Network Design and Modeling (ONDM)*, 2021, pp. 1–3.

[29] W. Jia, "A scalable multicast source routing architecture for data center networks," *IEEE Journal on Selected Areas in Communications*, vol. 32, no. 1, pp. 116–123, January 2014.

[30] G. Keller and M. M. T. Chakravarty, "Flattening trees," in *Euro-Par'98 Parallel Processing*, D. Pritchard and J. Reeve, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 1998, pp. 709–719.

[31] S. Routray *et al.*, "Statistical model for link lengths in optical transport networks," *J. Opt. Commun. Netw.*, vol. 5, no. 7, pp. 762–773, 2013.

[32] R. R. Fontes *et al.*, "Mininet-wifi: Emulating software-defined wireless networks," in *2015 11th International Conference on Network and Service Management (CNSM)*, 2015, pp. 384–389.

[33] I. Herstein, *Topics in Algebra*, 2nd ed. John Wiley & Sons, 1975.

Rafael S. Guimarães Currently holds an associate professor position in the Department of Informatics (DI) at the Federal Intitute of Espírito Santo (IFES), Campus Cachoeiro de Itapemirim, Brazil.

Cristina Dominicini Currently holds an associate professor position in the Department of Informatics (DI) at the Federal Intitute of Espírito Santo (IFES), Campus Serra, Brazil.

Victor M. G. Matinez is currently working towards a Ph.D. degree in Electrical Engineering at the Federal University of Espírito Santo, cooperating with the UFES' SDN Research Group.

Bruno M. Xavier is a Ph.D. student in computer science at UFES and associate professor at Federal Institute of Espírito Santo.

Diego R. Mafioletti is a Ph.D. student in computer science at UFES and associate professor at Federal Institute of Espírito Santo.

Ana C. Locateli Currently holds an associate professor position in the Department of Mathematics at the Federal University of Espírito Santo (UFES), Brazil.

Rodolfo Villaca Currently holds an associate professor position in the Department of Informatics (DI) at the Federal University of Espírito Santo (UFES), Brazil. His main research interests are in Computer Networks.

Magnos Martinello Currently holds an associate professor position in the Department of Informatics (DI) at the Federal University of Espírito Santo (UFES), Brazil. His main research interests are in Computer Networks.

Moisés R. N. Ribeiro Currently holds an associate professor position in the Department of Electrical Engineering of Federal University of Espírito Santo (UFES), Brazil.