PoT-PolKA: Let the Edge Control the Proof-of-Transit in Path-Aware Networks

Everson S. Borges [*†‡], Magnos Martinello [*§], Vitor B. Bonella [*], Abraão J. dos Santos [*], Roberta L. Gomes [*§], Cristina K. Dominicini [†], Rafael S. Guimarães [†§], Gabriel T. Menegueti [*], Marinho Barcellos [§], Marco Ruffini[‡]
[*]Department of Informatics, Federal University of Espírito Santo, Brazil.
Emails: {magnos.martinello, roberta.gomes, vitor.bonella, abraao.santos, gabriel.menegueti}@ufes.br,
[†]Department of Informatics, Federal Institute of Espírito Santo, Brazil
Emails: {everson, cristina.dominicini, rafaelg}@ifes.edu.br
[‡] Trinitty College Dublin, Ireland
Email: {marco.ruffini}@tcd.ie
[§] Waikato University, New Zealand
Email: {marinho.barcellos}@waikato.ac.nz

***Abstract*—This paper presents a scalable and efficient solution for secure network design that involves the selection and verification of network paths. The proposal addresses the challenges related to compliance policies by introducing a Proof-of-Transit (PoT) feasible implementation for path-aware programmable networks. Our approach relies on i) a source routing mechanism based on a fixed routeID representing a unique identifier per path, which serves as a key for PoT lookup tables; ii) the "in situ" that allows to collect telemetry information in the packet while the packet traverses a path. The former enables path selection with policy at the edge, while the later allows to perform path verification without extra probe-traffic. A P4 programmable language prototype demonstrates the effectiveness of this approach to protect against deviation attacks with low overhead. The results show its scalability considering the protocol overhead as the path length increases; a significant reduction in network's forwarding state for fat-tree topologies depending on the workload per path (flows/path). Finally, experimental results show a RTT comparison evaluation, the impact of PoT computation, protection to path deviation and seamless path migration keeping flow protection.**

I. Introduction

In the current Internet architecture, routers determine how a packet should be forwarded based on its destination. The forwarding decision relies on each router's local routing table. Each entry in the routing table associates a reachable destination with the next hop on the path. Unfortunately, in this architecture, there is almost no means for path verification [1], and an application can only assume that a packet will eventually reach the destination without selecting a specific path [2], opening up numerous failures possibilities. For example, an adversary may deviate the traffic violating the security policy, or a path can be invalid due to a Forwarding Information Base (FIB) corruption.

Internet Service Providers (ISPs) play a critical role in ensuring reliable data delivery. To maintain the highest level of service, ISPs must meet a Service Level Agreement (SLA). In today's rapidly evolving technological landscape, the demand for Network Function Virtualization (NFV [3]) and modern service chaining is increasing [4]. These new technologies require compliance with specific policies or regulations that specify the path that data must take through the network, including the specific nodes it must pass through. Additionally, ISPs must be able to prove that packets have passed through a set of service functions to ensure the delivery of accurate and secure data [5]. In short, ISPs must meet the SLA for data delivery in their network and comply with regulations to maintain the trust of their customers and remain competitive.

To meet these requirements, modern routing must have two properties: **path-awareness** [2] and **path-verifiability** [1]. Path-awareness allows endpoints to choose network paths by exposing path information at the network or transport layers. Path-verifiability provides Proof-of-Transit mechanisms to securely confirm that all packets within a given path passed through the intended nodes.

In this paper, we examine two concepts: (i) **Strict Source Routing (SSR)**, where a source node adds a route label in the packet header to specify all the nodes in an end-to-end path [6]; and (ii) **In situ Operations, Administration, and Maintenance (IOAM)**, which collects operational and telemetry information in the packet while the packet traverses a path [7]. The former enables path selection and reduces the control signaling and latency related to path setup [6], while the later allows to perform path verification without extra probe-traffic [5].

Our proposal, called PoT-PolKA, is a novel lightweight and scalable in-situ PoT approach for path-aware programmable networks that combines the SSR provided by **Pol**ynomial **K**ey-based **A**rchitecture (**PolKA**) [6] with a new version of the PoT mechanism introduced by a IETF RFC draft [5] based on the Shamir's secret sharing scheme [8].

As shown in Fig. 1, the first part of our design relies on the semantic of PolKA source routing that specifies a *routeID*, which is decoded at each node by a polynomial modulo operation for packet forwarding. This *routeID* expresses the

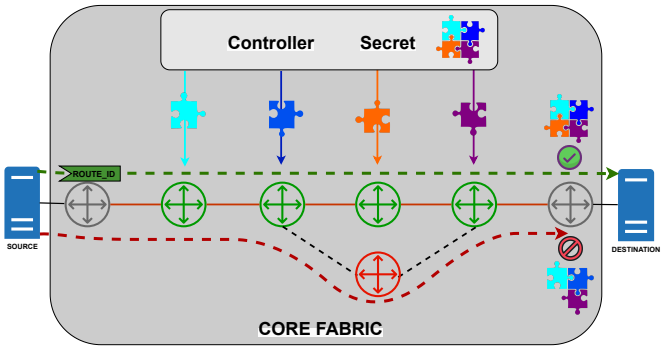


Fig. 1: PoT-PolKA Overview

entire path for the packet, i.e., not its destination address, but how to traverse each node until it reaches the destination. Moreover, this unique *routeID*, unchanged throughout the path, serves as a key for PoT table lookup to support the path verification.

The second part of our design is devoted to path-verifiability for which a shared secret is distributed by a Controller to the nodes in the path through a secure channel, as depicted in Fig. 1. At the ingress edge, metadata is added to in-situ header, and a cumulative number is updated at each hop. At the egress edge, the verifier node checks if the cumulative number in the packet header matches its secret.

In comparison to the IETF RFC Draft [5], this work offers a combined SSR and packet-path binding approach with **PolKA**, which allows aggregation of flows and avoids to store per-flow state on routers. This represents a major paradigm change, since the edge node[1] is now able to select route labels that represent not only paths, but also the PoT metadata associated to the respective paths. Therefore, this ability to bind all this information to a route label enables agile reconfiguration of paths and policies by changing a single entry at a source or edge node. Furthermore, our work introduces Mersenne numbers [9] for a feasible implementation in programmable switches of the modulo operations required by the Shamir mechanism. To the best of our knowledge, this is the first open-source implementation of PoT with Shamir mechanism using P4.

As proof-of-concept, a prototype is built for PoT-PolKA network (edge and core nodes) developed in P4 programmable language. For validation, PoT-PolKA scalability analysis is presented for different workloads, also presenting lower overhead even when the path length increases. For performance benchmarks, experimental results are carried out in Mininet emulated environment, showing PoT-PolKA low overhead to provide path migration keeping flow protection and protection against traffic deviation attacks.

II. Background and Related Works

This section describes the definitions, problem statement, background, and related works for our proposal.

[1]Depending on the context, the edge node may represented by an endpoint, such as a virtual switch or a smartNIC in a server, a hypervisor, a top-of-rack (ToR) switch, or a ingress domain gateway.

A. Definitions

Our proposal is built upon some important concepts that are defined as follows:

Proof-of-Transit (PoT): it is a security mechanism that uses cryptography for verifying the path through which a packet was forwarded [5], in contrast to conventional traceroute solutions. PoT is also known as path verification [1], which enables the destination to secure the metadata used to retrieve packet trajectories.

In-situ: it means that the Operations, Administration, and Maintenance (OAM) information is collected within the packet while the packet traverses a network domain, rather than send extra packets dedicated to OAM [7].

Path-aware networks: networks that expose path information and allow endpoints to select the specific path that the packet will traverse [2].

Programmable networks: networks that allow to decouple data and control plane configuration from network hardware by using application programming interfaces (APIs).

P4 language: it is a high-level language for programming the data plane of network devices in a protocol-independent manner [10]. Differently from traditional switches, the data plane functionality in P4 is not defined beforehand (e.g., match-action tables with fixed protocol columns), but is described by a program.

Source routing (SR): the routing path is determined at the source and the path computation only needs to be done once for each packet. The most common example of SR is Port Switching and this work adopts PolKA SR (more details in Section II-C) [6].

Secret Sharing Scheme (SSS): it is a method for dividing a secret into multiple shares or parts in such a way that the original secret can only be reconstructed after a certain number of these shares are combined. Examples of SSS are: SSSS (Shamir Secret Sharing Scheme), TSS (Threshold Secret Sharing), and LSSS (Linear Secret Sharing Schemes). In this proposal, we adopt the SSSS (see Section II-D).

Mersenne numbers: it is a class of integers closely tied to the field of number theory and prime numbers [9]. In Section II-D, we explain how we explore Mersenne numbers to enable the implementation of SSSS in P4 switches.

B. Problem Statement, Scope, and Security assumptions

In this paper, we want to answer the following research question: How to prove that a traffic flow follows the correct path for path-aware programmable networks? We address this research question by investigating the feasibility of a lightweight and scalable in-situ Proof-of-Transit approach. A path validation solution involves: (i) securing route propagation and authentication in the control plane (e.g., RKPI and BGPSec), and (ii) enforcing and verifying the correct transit of traffic in the data plane. The focus of our work is in the path verification, but it can benefit from several related works to implement a full path validation scheme in future works [1]. Path enforcement, route authentication and the discovery of path properties [2] are important security aspects [1], but they are out of the scope of this paper.

Our assumption for the adversary model is that a network attacker is able to deviate the traffic violating its security policy, which leads to *forwarding inconsistencies* (i.e., path deviation attacks) described in [1] as follows:

Skipping: Refers to a malicious routing behavior, wherein a router redirects a packet, intentionally bypassing one or more intermediary routers that are supposed to be part of the designated path. This behavior is depicted in Fig. 2, where, for instance, the packet avoids router 3 and proceeds along the alternative route, passing through the path $1 \rightarrow 2 \rightarrow 4 \rightarrow 5$.

Addition: The packet temporarily diverges from the originally intended path and subsequently rejoins it. During this process, the packet may traverse one or more routers that were not initially part of the expected route. A visual representation, in Fig.**??**, with the altered path: $1 \rightarrow 2 \rightarrow 6 \rightarrow 3 \rightarrow 4 \rightarrow 5$.

Path detour: Malicious router R_1 causes a packet to deviate from the intended path, but later it returns to the correct path. Partial detour is illustrated in Fig. 4, where the packet deviates from some but not all routers (path $1 \rightarrow 2 \rightarrow 6 \rightarrow 7 \rightarrow 4 \rightarrow 5$). Complete detour is shown in Fig. 5.

Fig. 2: Skipping

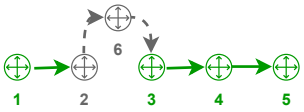


Fig. 3: Addiction

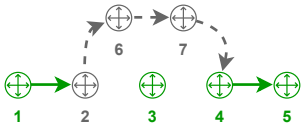


Fig. 4: Partial Detour

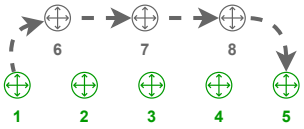


Fig. 5: Complete Detour

PoT prevents forgery and adulteration; an attacker cannot maliciously create a fake PoT or modify a legitimate one. Furthermore, a legitimate node that is part of the PoT protocol cannot disguise itself as another node along the way. Unfortunately, no scheme can provide any guarantees for malicious routers: if malicious router R_i publishes its secret key, another malicious router R_i' could perform cryptographic operations on a packet without traversing R_i. Besides, under the control of attackers, compromised routers may discard or detour packets. Moreover, the PoT mechanism discussed in this paper does not verify the order in which nodes were traversed, i.e., it does not detect where packets are being drifted from the selected path. However, it can be extended to enable Ordered Proof of Transit with symmetric masking between the nodes, as described in [11]. Also, other performance metrics such as the per-hop delay, and the load of each network interface might be included in future work.

We envision our proposal can be applied in different contexts where the information regarding the underlying network is available: (i) networks that are owned and managed by the same operator, as datacenter networks; (ii) networks that share information about their interior topology with each other in some type of trust relationship; or (iii) application-layer services like ALTO [12], which use such information to perform better-than-random selection of the endpoints.

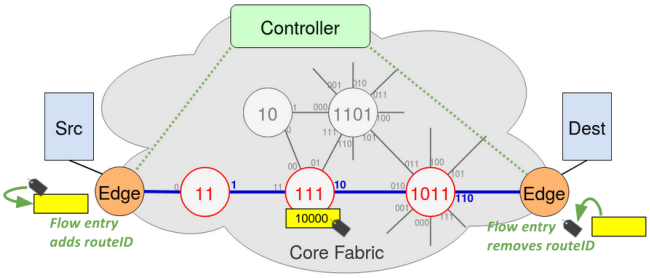


Fig. 6: Example of PolKA SR.

A practical application is the problem of detecting and troubleshooting Equal-Cost Multi-Path (ECMP) [13] paths. This problem remains a big challenge that faces many operators [14]. ECMP typically distributes traffic to all available paths in a round-robin fashion, but FIB corruption or FIB misconfiguration at any node is one among many issues that can happen and may lead packets to follow a non ECMP Path. Ideally, a PoT allows to detect the currently available ECMP paths, being able also to know whether packets are following an ECMP Path or not.

C. PolKA Source Routing

Related works investigated the benefits of SR over traditional table-based routing, such as reduction of network states and optimal use of network capacity [15]. The most traditional way of executing SR is Port Switching, in which the route label represents a ordered list of output ports (or network addresses) and each hop executes the forwarding operation as a pop of the first element of the list [15]. As a consequence, Port Switching needs to update the route label in the packet for each hop.

On the other hand, PolKA SR explores the residue number system (RNS) with Galois field (GF) polynomials of order 2 to derive the route label, which remains unchanged throughout all the path [6]. In this scheme, at any core node, the output port (*portID*) is given by the remainder of the binary polynomial division (i.e., a mod operation) of the route identifier (*routeID*) of the packet by the node identifier (*nodeID*). The polynomial mod operation is enabled in programmable switches by the reuse of the cyclic redundancy check (CRC) hardware [6].

This section presents the necessary background to understand how PolKA is used as the routing method of PoT-PolKA to bind paths to its respective PoT metadata. Since the route label in PolKA remains the same along the path, it can be used as an index for table lookups, allowing flow aggregation and reducing per-flow state in the network.

Consider 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 *nodeID*s 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 [6] as the polynomial $R(t)$ that satisfies:

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

The algorithm complexity for computing $R(t)$ is $O\left(len(M)^2\right)$ [6], where $M(t) = \prod_{i=1}^{N} s_i(t)$. The *routeID* is embedded in the packet by the edge, and 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. 6 shows an example for a path composed of three core nodes, which received their *nodeID*s 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 output port set O is composed by the polynomials: $o_1(t) = 1, o_2(t) = t = 10, o_3(t) = t^2 + t = 110$. 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).

D. Shamir's Secret Sharing Scheme and Mersenne numbers

SSSS provides a secure and efficient threshold method based on polynomial interpolation over finite fields (using the Lagrange Interpolation Formula [16]) for sharing a secret among a group of participants [8]. Firstly, in a share distribution phase, it distributes a limited number of n shares. Then, in a secret reconstruction phase, it requires at least the threshold number of t shares to reconstruct the secret.

In practice, a secure secret sharing scheme is one where individuals possessing fewer than (t) shares have no advantage over those with zero shares in terms of their knowledge about the secret. SSSS adheres to this criterion and achieves information-theoretical security, meaning that it cannot be compromised by an attacker, even with unlimited computational resources [5]. On the other hand, TSS represents a more simplistic form of secret sharing [8], where the threshold (t) equals the total number of participating parties, resulting in the secret being fully disclosed to every party. This approach, although straightforward, lacks security since any compromised part immediately exposes a portion of the secret. Consequently, TSS is generally considered unsafe for safeguarding confidential information [17].

Although the basic PoT problem could be theoretically solved with a more simplistic secret sharing scheme if all the nodes are selected (i.e., $t = n$), we envision to exploit the particular properties of SSSS to leverage advanced PoT for probabilistic telemetry. The reasoning is that existing telemetry techniques incur high overheads due to requiring perfect telemetry information, but most applications do not required to know all of the per-packet-per-hop information that INT collects [18]. Thus, in future works, we plan to extend this current PoT proposal as follows: (i) for each packet of a flow, collect telemetry information of a random subset of any t nodes from the n nodes that compose a path; (ii) use SSSS to prove that the packet correctly transversed t nodes of the selected path; and (iii) reconstruct the probabilistic telemetry information to prove sufficient network coverage.

Therefore, considering its superior security robustness and flexibility, SSSS was selected as the best candidate to enable a broader PoT solution. Nonetheless, SSSS algorithm requires integer modulo operations that may not be natively supported on the data plane of current programmable switches. To enable a feasible implementation of SSSS using the P4 language, we propose a technique to calculate the modulo operation with Mersenne numbers [9], which follow the pattern $(2^B) - 1$.

A number modulo a Mersenne number can be calculated by a shift bitwise operation (<<). Algorithm 1 shows how the calculation of $k \ mod \ p$ can be executed with elementary operations, where p is a Mersenne number, k is smaller than $(1 << (2 * B)) - 1$, and B is the power of two of the Mersenne number $((2^B) - 1)$ [2]. To clarify, assuming that $p = 31$ $B = 5$, it means that k should be smaller than $(1 << 10) - 1 = 10000000000_2 - 1_2 = 1024 - 1 = 1023$.

Algorithm 1: Modulo Mersenne Number

Data:
k - Input integer
p - Integer value
B - Integer value
Result:
i - Integer value

```
i ← (k&p) + (k ≫ B);
if i ≥ p then
    i ← i − p;
return i;
```

E. Related Works

In this section, we review the literature considering the related works in Proof-of-Transit (PoT), but also the recent efforts towards PoT standardization.

Regarding the PoT related works, ICING [19] relies on aggregate message authentication codes (MAC [20]) and self-certifying names to enforce path consent and path compliance. Despite its linear network state with respect to path length, overhead is high with a proof of consent from the nodes in the path. Besides, its verification needs a variable header stack (list of on-path nodes), imposing complexity for implementation.

[2] https://ariya.io/2007/02/modulus-with-mersenne-prime

Different from ICING, OPT [21] does not include the list of on-path nodes in the packet header. OPT assumes that all nodes trust the source S, and each on-path node N_i generates a shared symmetric key k_i with S. OPT refers to secrets as origin and path validation (OPV); it allocates one OPV_i field for each N_i in the packet header with 128 bits each. Although it imposes less complexity compared to [19], it also requires variable header size (on-path OPV_i). Extended-OPT [22] is a variant of OPT that suggests to keep the same complexity when nodes do not trust the source.

Orthogonal sequence verification (OSV) [23] follows the same design principle as OPT, but with lighter orthogonal sequences. OSV also relies on a path validation field (PVF) and an original validation field (OVF) per on-path node, but achieves faster computation of those fields. PPV [24] takes a different approach with probabilistic path validation. Its premise is that each packet does not need to be marked by all of the routers it visits (at most two routers along the forwarding path). It is based on per-flow path validation, so that PPV routers only mark packets with a certain probability.

In terms of stardardization efforts, there is a recent RFC in IETF for IOAM deployment [7], that allows to use a variety of data header fields (RFC 9197 [25]). More specifically to POT, the IETF-draft [5] explores the Shamir's secret sharing scheme [8] to provide an in-situ PoT solution. The main advantage with respect to the previous works is its lightweight in-situ fingerprint [7] with a small cumulative signature and low control plane overhead. However, this IETF PoT proposal [5] has some drawbacks: (i) it depends on an integer modulo operation that is not commonly supported in programmable switches; (ii) if it uses traditional table-based routing, it still requires large numbers of table entries, which restricts path selection [6]; and (iii) it aggregates PoT policies per flow, impacting the scalability of the PoT solution.

Table I shows a system design comparison, where the first line refers to PoT IETF-draft [5] which depends on table-based routing and has no path awareness. An alternative is to replace table-based routing by Segment Routing [26] (line 2 of the Table I) in order to allow path selection, but, since it represents the path as a list of nodes and updates this list on each hop, it leads to a variable size header.

PoT-PolKA is presented as another design choice in Table I. It is built on the same basis as the PoT IETF-draft, maintaining the advantages related to the use of SSSS (lightweight in-situ fingerprint and low control plane overhead). On the other hand, it introduces the use of source routing in combination with PoT, which enables the edge to select the paths and bind them to PoT metadata. Moreover, it has capabilities that are unique compared to existing works: **i)** encoded path information; **ii)** constant route identifier; **iii)** aggregation per path not per flow; and **iv)** *routeID* overhead for the data plane, instead of path tracing or variable size header. Besides, PoT-PolKA offers a feasible implementation of the modulo operation with Mersenne numbers that allows the deployment on P4 programmable switches.

III. PoT-PolKA proposal

This section introduces PoT-PolKA proposal, presenting a comparative overview with the existing IETF RFC in section III-A. In subsection III-B, a step-by-step design is described with its implementation in P4 code.

A. Overview

The Fig. 7 presents the overview of IETF RFC Draft [5] based on the Shamir secret sharing [8] method. The system parameters are provisioned by the controller and header metadata is updated at every hop. At the egress node, the collected meta-data allows to reconstruction of the secret for path verification. Thus, PoT metadata ($rnd, cml = 0$) is inserted into the packet header at the edge. In node A, the PoT table is checked in order to update the PoT metadata and its respective routing table to forward the packet to the output link. The packet in transit has its PoT metadata updated with its path tracing ($A, link_1$). Then in node C, the process is repeated until the egress edge node, updating the PoT metadata ($rnd, cml = 44$) and stacking its path tracing ($B, link_4$). The path verification is performed at the egress edge that checks whether the collected meta-data matches with the cumulative PoT metadata ($rnd, cml = 55$). It is important to note that each core node stores tables for routing and PoT parameters.

On the other hand, PoT-PolKA design offers a path-binding property by using a SSR approach based on PolKA [6], which explores the Residue Number System (RNS) and Chinese Remainder Theorem (CRT). PolKA encodes the path in a *routeID* Fig. 8 using the RNS in contrast to the conventional table-based, which depends on routing tables, or list-based representations, which transports the path information "in clear" inside the packet header. Then, PolKA core nodes use this encoded route label to discover the output ports, by performing the forwarding as an arithmetic operation: the remainder of division of the *routeID* by its own *nodeID*. However, if a FIB corruption occurs e.g. due to a fault injection by an attacker, causing the remainder of division to forward the packet to another output port, it would lead to path deviation attacks. Thus, assuming she/he gets access to *nodeID* and *portID*, despite PolKA first security barrier, this will not guarantee the forwarding consistency, so that a PoT is required to protect against packet path deviation.

PoT-PolKA solves problems in traditional PoT solutions by proposing a design based on Shamir's secret sharing scheme and PolKA. It uses programmable P4 switches and a small packet digest (PoT metadata) to ensure the path-binding property. The PolKA *routeID* acts as a key to check the nodes along the defined path and update the PoT metadata at each hop. The egress node verifies if the packet traversed all the specified nodes without the need for storing path tracing information, as the *routeID* uniquely identifies the path.

Limitations of the approach and additional security analysis can be seen in [5], with proofs of robustness for *inter-node* and *inter-packets* passive attacks. However, the current solution does not mitigate *replay* and *pre-replay* attacks, requiring a mitigation mechanism to be included in future versions.

Table I: PoT system design comparison

Method	Routing	Path Info	Route Identifier	Policy	Control Plane Overhead	Data Plane Overhead	Imp. Prog. Switches
PoT IETF-draft [5]	Table-Based Routing	In clear	None	Per flow	Routing tables PoT configs	Path tracing PoT metadata	No
PoT IETF-Draft deployed with Segment Routing	List-based Source Routing	In clear	Variable	Per flow	PoT configs	Path tracing PoT metadata	No
PoT-PolKA	PolKA Source Routing	Encoded	Constant	Per path	PoT configs	PoT metadata	Yes

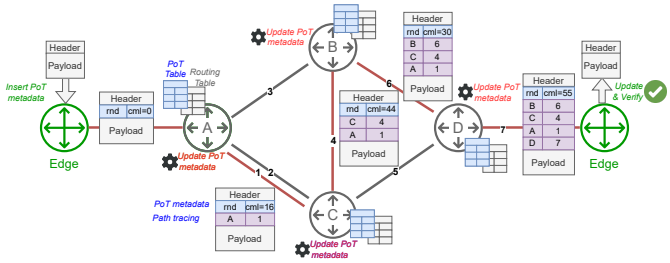


Fig. 7: PoT proposal by IETF Draft

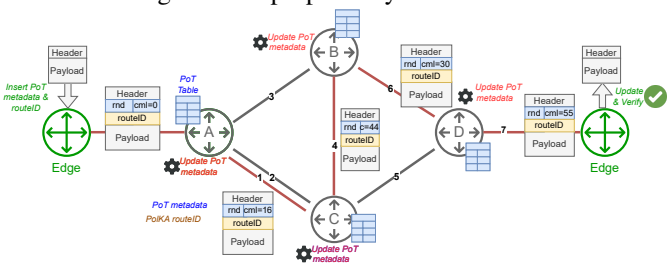


Fig. 8: General Design of PoT-PolKA

B. Design

Fig. 9 explains the PoT-PolKA step-by-step. The design is structured in three main steps: the computations at control plane, configuration of the data plane, and, finally, the path verification in the egress edge node.

1) Step1 : PoT Computation at the control plane: The PoT-PolKA algorithm leverages on Shamir's Secret Sharing scheme [8]. The principle is to define a single secret, represented by a polynomial, that is associated with a particular set of $n+1$ nodes that typically represent the path to be verified. Thus, a polynomial of degree n is selected as a secret at the control plane. A set of $n+1$ points of this polynomial will be assigned to $n+1$ nodes. Each of these $n+1$ points is called a "share" of the secret.

For the **edge nodes**, a private polynomial ($Poly_1$) is selected (see fig. 9), and its zero degree coefficient gives the secret (e.g. $Secret = 10$). When a path is selected to be verified, for each pair of edge nodes, $(x, Poly_1(x)\ mod\ M)$ must be computed. For example, see the green box ($P1 = 16, M = 31, B = 5, S = 10$). Note that the parameters B and M refer to the introduction of Mersenne numbers for a feasible implementation of SSSS in programmable switches and represent a novelty in comparison to the IETF RFC Draft [5].

For the **core nodes**, a public polynomial ($Poly_2$), as there are $n+1$ nodes in the path, the polynomials ($Poly_1$,($Poly_2$)) should be of degree n, is chosen and the verifier egress node can reconstruct the n degree polynomial ($Poly_3$) only when all the points are correctly retrieved. The shares of the secret are the points on ($Poly_1$) chosen for a path length of 4 nodes. For example:

$x_0 = 1,\ x_1 = 3,\ x_2 = 5,\ x_3 = 7$

$Poly_1(1) = 16 = (x_0, y_0) = (1,\ 16)$

$Poly_1(3) = 15 = (x_1, y_1) = (3,\ 15)$

$Poly_1(5) = 7 = (x_2, y_2) = (5,\ 7)$

$Poly_1(7) = 23 = (x_3, y_3) = (7,\ 23)$

Lagrange polynomial interpolation is used for secret reconstruction to a given set of points on the curve [5]. The Lagrange Polynomial Constants ($LPC's$) [16] are computed by the Controller and communicated to the nodes. Since the points are $x_0 = 1$, $x_1 = 3$, $x_2 = 5$, $x_3 = 7$ in the example, ($LPC's$) can be computed as follows:

$$LPC(x_0) = \frac{0-x_1}{x_0-x_1} * \frac{0-x_2}{x_0-x_2} * \frac{0-x_3}{x_0-x_3} = \frac{105}{48}\ mod\ 31 = 8$$

$$LPC(x_1) = \frac{0-x_0}{x_1-x_0} * \frac{0-x_2}{x_1-x_2} * \frac{0-x_3}{x_1-x_3} = \frac{35}{16}\ mod\ 31 = 23$$

$$LPC(x_2) = \frac{0-x_0}{x_2-x_0} * \frac{0-x_1}{x_2-x_1} * \frac{0-x_3}{x_2-x_3} = -\frac{21}{16}\ mod\ 31 = 11$$

$$LPC(x_3) = \frac{0-x_0}{x_3-x_0} * \frac{0-x_1}{x_3-x_1} * \frac{0-x_2}{x_3-x_2} = \frac{45}{48}\ mod\ 31 = 21$$

2) Step 2 : Data plane configuration: In this stage, the parameters are assigned to a PoT table at the nodes. According to PolKA routing [6], the *routeID* is the key (e.g., $routeID$ = R1 = 10979360238159862843) needed to perform the actions in the table. Also, the *nodeID*s are generated and associated to these nodes in the path ($Core_{Node}(1)$ = 65579, $Core_{Node}(2)$ = 65581, $Core_{Node}(3)$ = 65593, $Core_{Node}(4)$ = 65599)[3].

It is worth noting that each parameter is kept secret by individual nodes (i.e. precisely the points on $Poly_1$, the share of $Poly_2$, LPC, M, B). Only the constant coefficient (RND) of $Poly_2$ is public, whereas x value and non-constant coefficient of $Poly_2$ are secret.

On the edge (green table), they receive the information about the secret ($Secret$) and the fixed polynomial ($Poly_1$). The core nodes receive respectively the pair (X, Y) and the LPC of the node, and the polynomial ($Poly_2$). There are some conditions to choose the polynomials: Assuming that K_1 is the degree of $poly_1$, and K_2 is the degree of $poly_2$ with N nodes in the core, we need $K_1 < N$ and $K_2 < N$. Thus, as we use

[3]PolKA project with github implementation and examples can be found at: https://github.com/nerds-ufes/polka/tree/main/mininet/pot-polka

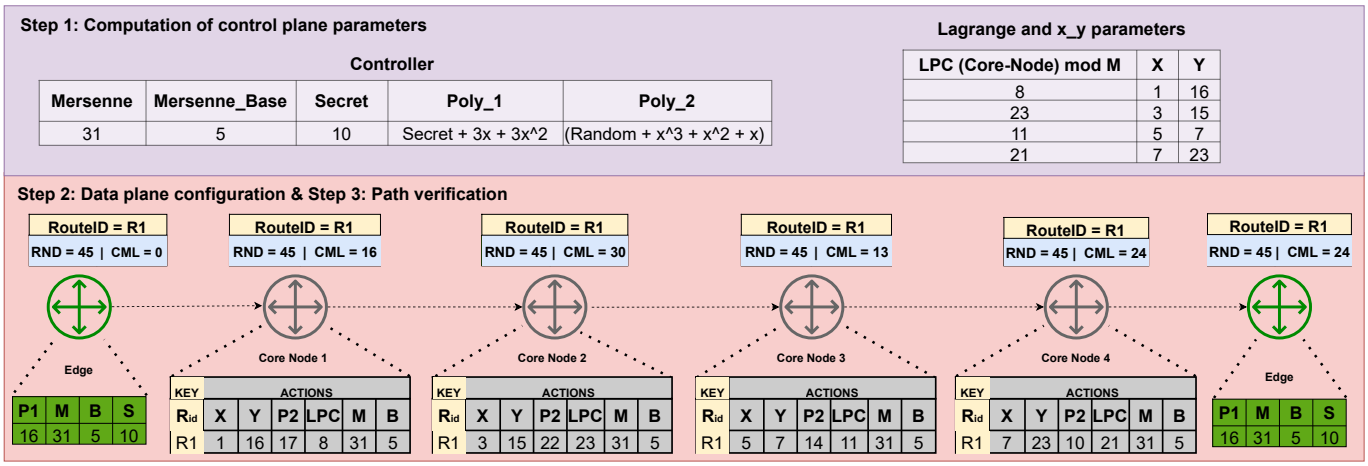


Fig. 9: PoT-PolKA design step-by-step

polynomials of minimum degree equal 2, the number of core nodes must be at least 3. In Figure 7, $poly_1$ has degree 2 and $poly_2$ has degree 3, so the minimum number of core nodes must be 4. Finally, the Mersenne M with B is assigned to all nodes.

Data plane computation: In operation, each packet carries its PoT metadata with a random value (RND), generated by the edge, and a cumulative of secret (CML) that is initially zero. The CML is updated by every core node by computing the current CML with the Equation 2, which is implemented in the P4 language (as detailed in Code 1):

```
action calc_cml(){
    meta.new_cml = (meta.y + meta.poly2) * meta.lpc;
    meta.new_cml = (meta.new_cml & meta.mersenne) +
        (meta.new_cml >> meta.mersenne_b);
    if (meta.new_cml > meta.mersenne){
        meta.new_cml = meta.new_cml - meta.mersenne;
    }
    meta.new_cml = hdr.potPolka.cml + meta.new_cml;
}
apply { PoT-PolKA pipeline
    if (hdr.potPolka.isValid()){
        // Calculate egress port using PolKA SR
        srcRoute_nhop();
        // Table lookup to initialize PoT parameters
        pot_param.apply();
        // Calculate and update CML
        calc_cml();
        hdr.potPolka.cml = meta.new_cml;
        // Set egress port
        standard_metadata.egress_spec = meta.port;
    }else{drop();}
}
```

Code 1: PoT-PolKA Data Plane Computation in P4 Code

$$CML = (CML + (Poly_1(X) + Poly_2(X)) * LPC) \ mod \ M \quad (2)$$

3) Step 3: Path verification: In the verifier node, the verification is made by comparing if the CML in the packet header and the $VERIFY$ value are equal (Equation 3):

$$VERIFY = (S + RND) \ mod \ M \quad (3)$$

As can be seen in Fig. 9, the RND remains fixed during the path, but the CML is computed hop by hop. So, in the egress edge, the PoT applies the equation 3. Given that the $VERIFY = 24$ computed in the edge is equal to $CML = 24$ in the packet header, the path verification is confirmed.

IV. Evaluation

The assessment methodology is structured in two parts. Firstly, we focus on the scalability analysis (section IV-A) by i) considering the protocol overhead in ICING [19] and OPT [21] versus PoT-PolKA as the path length increases ; ii) evaluating the reduction on network states achieved by PoT-PolKA versus PoT-IETF. The comparison deals with fat tree topologies with different sizes and varying the workload in terms of flows per path.

For the second part of the evaluation (section IV-B), a prototype of PoT-PolKA was developed as Proof-of-the-Concept (PoC) and experiments were conducted within the Mininet emulation platform. We start the experimental evaluation by a comparative analysis measuring the RTT performed among Baseline, PoT-PolKA, and PoT-IETF. The impact of PoT computation is evaluated by the additional latency introduced by PoT-PolKA compared to PolKA, as the number of hops on the path increases. Then an experiment is devoted to demonstrate the traffic deviation within the network, which may potentially violate security policies, and PoT-PolKA's protection mechanism being able to detect the path deviation (invalid path). The last experiment was designed to show a seamless path migration keeping the flow protection under the change of network conditions.

A. Scalability analysis

Path length

In order to analyze the protocols' scalability as a function of the path length, we take the same values from [21] of 256B and 1024B packets size, varying the path length from 2 to 10 hops. Fig. 10 shows the comparison result. The increase on path length adds more overhead to the packet header, for both OPT and ICING. However, as expected, the PoT-PolKA header remains the same with lower overhead than ICING and OPT as the path length increases.

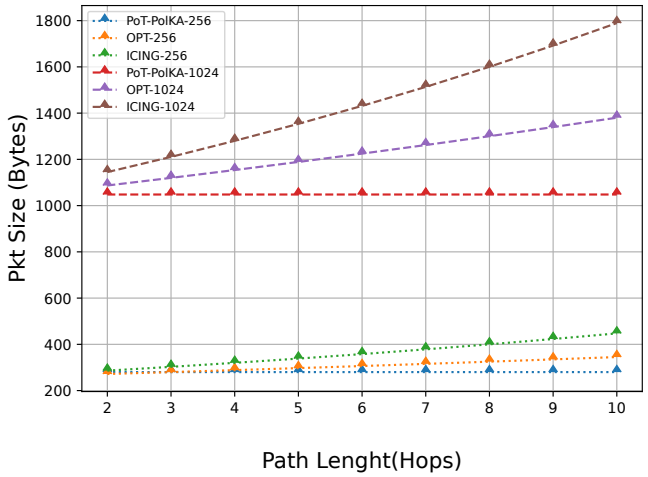


Fig. 10: Packet size for OPT, ICING, and PoT-PolKA.

Network state

For evaluation purposes, we assume that each rule (i.e. network state) is a flow entry for exact matching at the PoT table. So, for the IETF RFC draft, we have $N = flows\_per\_path * number\_of\_paths * path\_length * 2$. For PoT-PolKA, as it aggregates multiple flows that cross a path avoiding to store per-flow state on routers, then the number of network states is $N = number\_of\_paths * path\_length * 2$.

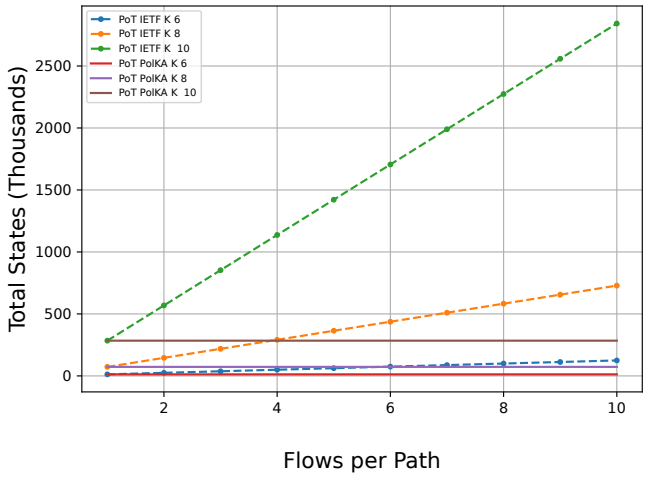


Fig. 11: Number of states for Fat-Tree topologies.

Fig. 11 presents a network state requirements comparison between PoT-PolKA and the IETF draft PoT. We use fat-tree topologies for different $K = 6, 8, 10$ under a variable workload (flows per path) from 1 to 10. The pod path lengths were calculated for all combinations of nodes, either for *intra* or *inter* pod. As can be seen, the heavier is the workload per path, the greater is the reduction achieved by PolKA on the total number of states. For example, for a fat-tree with $K = 10$ and a workload of 6 flows per path, the reduction achieves 83,3% and 90% for 10 flows per path.

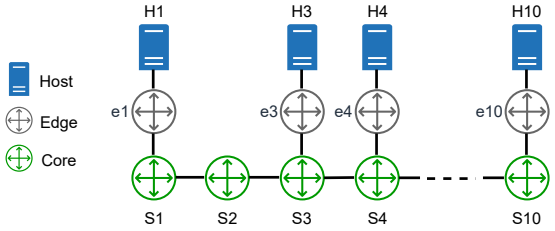


Fig. 12: Linear Fabric Topolology

B. Experimental Evaluation

To evaluate the main functionalities of PoT-PolKA, we developed a prototype in the software switch *bmv2 simple_switch* with the v1model architecture as the target. The scenario of Fig. 12 shows a linear fabric topology with edge and core nodes emulated in Mininet. The main objective is to have an experimental PoC for PoT-PolKA validation. The experiments´ goal is to compare the overhead of PoT-PolKA proposal to the previous works, as the number of hops increases in the core network (e.g., from 3 hops for path H1 $\rightarrow$ H3 to 9 for path H1 $\rightarrow$ H10).

The physical setup consists of a server Dell PowerEdge T430, with an Intel Xeon E5-2620 v3 2.40GHz processor and 64GB of RAM. We ran experiments within an Ubuntu 18.04.6 LTS. To build our emulated environment, we used Mininet 2.6 with a P4 compiler and *bmv2* 1.15.0.

RTT comparison

Fig. 13 shows a RTT comparison among a Baseline, PolKA, PoT-PolKA, and PoT-IETF. Specifically, the table-based packet forwarding is used as a baseline. PoT-PolKA represents our solution applied to the linear topology varying the path length (Fig. 12). Conversely, PoT-IETF was devised using a table that incorporates the PoT mimicking the IETF draft [5]. As can be seen in Fig. 13, there is a linear growth with small variation on RTT. Notably, PoT-IETF shows a slight disadvantage in comparison to PoT-PolKA due to the effect of lookup table delay and PoT computation, while PoT-PolKA increases RTT by 12% compared to Baseline due to its PoT computation.

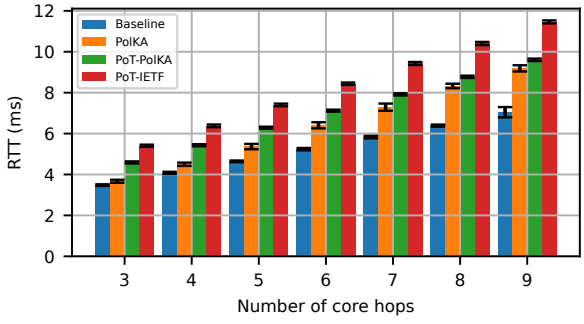


Fig. 13: RTT for Baseline, PolKA, PoT-PolKA and PoT-IETF

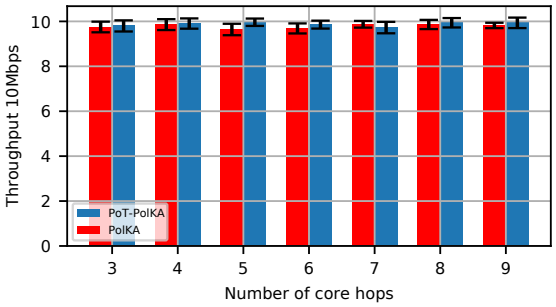


Fig. 14: Throughput between PolKA and PoT-PolKA

Impact of PoT computation

Comparing the PolKA vs. PoT-PolKA, we observe a small increase on latency by PoT-PolKA (around 4% when the path length is longer than 6). Throughput is essentially the same for both (Fig. 14), although it is worth mentioning that this is a relative value because the link rates were limited to 10 Mbps, due to *bmv2 simple_switch* processing capacity in the mininet emulation.

Protection to path deviation

In Fig. 15, a network attacker is able to deviate the traffic, violating the security policy. By using the prototype of PoT-PolKA in the Mininet emulation, two flows are created at *src* host: (i) a green line flow crossing $S1 \rightarrow S2 \rightarrow S3 \rightarrow S4$ (from T=0s to T=40s); and (ii) a red line flow, deviated from the intended path by the attacker, going over $S1 \rightarrow S5 \rightarrow S4$ (from T=20s to T=40s).

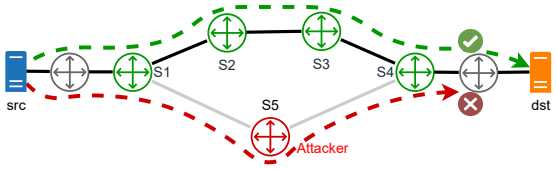


Fig. 15: Path deviation attack

As shown in Fig. 16, at T=20s, a new flow of 3Mbps is initialized and the aggregated flows at *src* (blue line) increase from 4Mbps to 7Mbps. However, this new flow was deviated which leads to the red line at 3Mbps, whereas the orange line representing the *dst* host remains at 4Mbps. Since the egress edge node applied the PoT-PolKA verification, it drops 3Mbps of the aggregated flows, demonstrating the PoT-PolKA protection against the path deviation attack.

Seamless path migration keeping flow protection

The purpose of this experiment is to provide a practical illustration demonstrating the traffic engineering support for PoT-PolKA while keeping the flow verification. In Fig. 17, two paths are established from H1 to H2. Path 1 comprises the following sequence of nodes: E1 - S1 - S2 - S3 - S4 - E2.

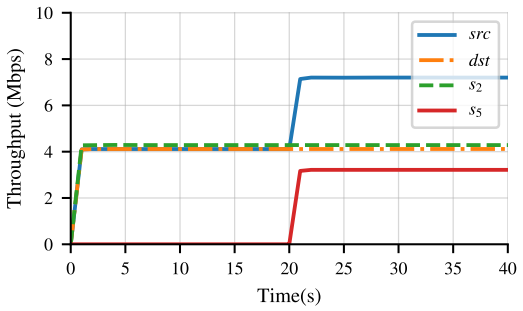


Fig. 16: Throughput with PoT-PolKA protection

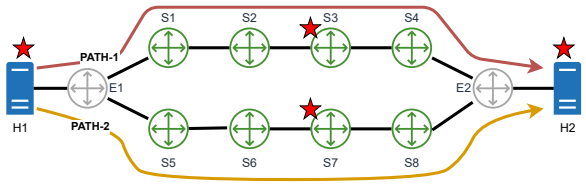


Fig. 17: Topology for Path Migration

Conversely, Path 2 is constructed using the following sequence of nodes: E1 - S5 - S6 - S7 - S8 - E2. The stars assigned to the nodes in the fig. 17 are the measurement points to collect the traffic so that we can plot the throughput from the switch perspective.

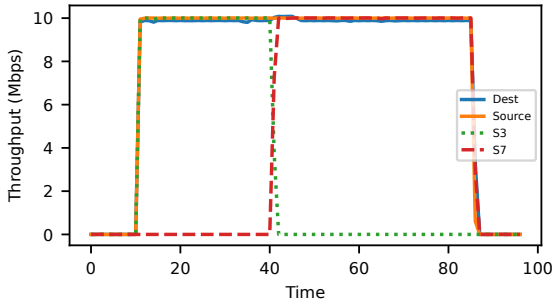


Fig. 18: Path Migration

In Fig. 18, we present the results of a path migration scenario. Initially, an UDP flow was initiated from source H1 to destination H2, following path 1. The traffic crosses through switch S3 (green), which is a node on path 1, and no traffic is crossing S7 (red). However, after 30 seconds, a change in the flow path occurs at the network E1, resulting in a modification of the *routeID*, which requires only an update at the edge to map the flow to this new *routeID*. Thus, the flow diverts away from S3 traversing now S7 (a measurement node on path 2). Interestingly, for this case of low flow rate, no packet loss was observed. This was due to the fact that path migration only requires an update at the network edge, as opposed to updates for all the switch tables along the path. Particularly noteworthy is that the verification mechanism is tailored to each *routeID* (designed per *routeID*), ensuring that this new path corresponds to a valid route, allowing this flow to seamlessly migrate reaching its destination.

A final remark for this experiment, the proactive instrumentation of switches for both paths, along with their respective route-ids (which is the key value for the PoT switch table), means that convergence time solely entails updating only at the edge, remaining completely transparent. Nonetheless, if any route-ids along the path had not been previously instrumented at the switches, an update (at the switch PoT table(s)) would have been necessary, resulting in additional time required for routing convergence.

V. Conclusion

This paper introduces a novel PoT design for programmable networks that leverages PolKA source routing [6] for strict path selection and a modified version of PoT IETF RFC draft [5] for path verification. The design integrates seamlessly with P4 programmable switches, resulting in a scalable and efficient implementation. The proposed solution is validated through experiments performed using a software switch implementation and the Mininet emulation platform, evaluating metrics such as RTT and throughput.

The proposal extends the IETF RFC draft [5], by introducing the Mersenne numbers for a feasible implementation of Shamir Secret Sharing in programmable switches. This is enabled by a fixed *routeID* that represents a unique path in the administrative domain and is used as a key for the PoT lookup table to support the path verification. Thus, we can aggregate PoT policies by path with a reduction on the number of network states. Furthermore, the PoT-PolKA header consistently maintains its efficiency advantage over ICING and OPT, demonstrating lower overhead as path length increases.

We envision as future work to devote efforts to deploy our approach at P4 Tofino programmable switches [27], to include path enforcement and validation functionalities, to explore PoT with probabilistic telemetry, and to extend [28] for secure multipath routing.

VI. Acknowledgments

Financial support from Brazilian agencies: CNPq, CAPES, FAPESP/MCTI/CGI.br (PORVIR-5G 20/05182-3, and SAWI 20/05174-0), FAPES (94/2017, 281/2019, 515/2021, 284/2021, 06/2022, 1026/2022, 941/2022). CNPq fellows Dr. Martinello 306225/2020-4. SFI 13/RC/2077_p2 and 17/CDA/4760. This work also received funds from the 2021 Google Research Scholar Award and the 2022 Intel Fast Forward Initiative.

References

[1] K. Bu *et al.*, "Unveiling the mystery of internet packet forwarding: A survey of network path validation," *ACM Computing Surveys (CSUR)*, vol. 53, no. 5, pp. 1–34, 2020.

[2] B. Trammell, "Current open questions in path-aware networking," IRTF, RFC 9217, Mar 2022. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc9217

[3] R. Mijumbi, J. Serrat, J.-L. Gorricho, N. Bouten, F. De Turck, and R. Boutaba, "Network function virtualization: State-of-the-art and research challenges," *IEEE Communications surveys & tutorials*, vol. 18, no. 1, pp. 236–262, 2015.

[4] P. Quinn, U. Elzur, and C. Pignataro, "Network service header (NSH)," IETF, RFC 8300, Jan 2018. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc8300

[5] F. Brockners, S. Bhandari, T. Mizrahi, S. Dara, and S. Youell, "Proof of Transit," Internet Engineering Task Force, Internet-Draft draft-ietf-sfc-proof-of-transit-08, Nov. 2020, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/draft-ietf-sfc-proof-of-transit/08/

[6] C. Dominicini, D. Mafioletti, A. C. Locateli, R. Villaca, M. Martinello, M. Ribeiro, and A. Gorodnik, "Polka: Polynomial key-based architecture for source routing in network fabrics," in *2020 6th IEEE Conference on Network Softwarization (NetSoft)*. IEEE, 2020, pp. 326–334.

[7] F. Brockners, S. Bhandari, D. Bernier, and T. Mizrahi, "In Situ Operations, Administration, and Maintenance (IOAM) Deployment," RFC 9378, Apr. 2023. [Online]. Available: https://www.rfc-editor.org/info/rfc9378

[8] A. Shamir, "How to share a secret," *Commun. ACM*, vol. 22, no. 11, p. 612–613, nov 1979.

[9] B. Tuckerman, "The 24th mersenne prime," *Proceedings of the National Academy of Sciences*, vol. 68, no. 10, pp. 2319–2320, 1971.

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

[11] A. Aguado, D. R. Lopez, V. Lopez, F. de la Iglesia, A. Pastor, M. Peev, W. Amaya, F. Martin, C. Abellan, and V. Martin, "Quantum technologies in support for 5g services: Ordered proof-of-transit," in *45th European Conference on Optical Communication (ECOC 2019)*, 2019, pp. 1–3.

[12] J. Seedorf and E. Burger, "Application-Layer Traffic Optimization (ALTO) Problem Statement," RFC 5693, Oct. 2009. [Online]. Available: https://www.rfc-editor.org/info/rfc5693

[13] C. Hopps and D. Thaler, "Multipath Issues in Unicast and Multicast Next-Hop Selection," RFC 2991, Nov. 2000. [Online]. Available: https://www.rfc-editor.org/info/rfc2991

[14] M. Valentine, "Nanog presentation." [Online]. Available: https://www.segment-routing.net/path-tracing/2022-06-08-NANOG85-path-tracing/

[15] 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.

[16] D. Quadling, "Lagrange's interpolation formula," *The Mathematical Gazette*, vol. 50, no. 374, pp. 372–375, 1966.

[17] M. Rosulek. (2021) The joy of cryptography. [Online]. Available: https://joyofcryptography.com

[18] R. Ben Basat *et al.*, "Pint: Probabilistic in-band network telemetry," in *Proceedings of the 2020 ACM Conference on SIGCOMM*, ser. SIGCOMM '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 662–680.

[19] J. Naous, M. Walfish, A. Nicolosi, D. Mazieres, M. Miller, and A. Seehra, "Verifying and enforcing network paths with icing," in *Proceedings of the Seventh Conference on Emerging Networking Experiments and Technologies*, 2011, pp. 1–12.

[20] J. Katz and A. Y. Lindell, "Aggregate message authentication codes," in *Topics in Cryptology – CT-RSA 2008*, T. Malkin, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 155–169.

[21] "Lightweight source authentication and path validation," in *Proceedings of the 2014 ACM Conference on SIGCOMM*, ser. SIGCOMM '14. New York, NY, USA: ACM, 2014, p. 271–282.

[22] F. Zhang *et al.*, "Mechanized network origin and path authenticity proofs," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '14. New York, NY, USA: ACM, 2014, p. 346–357.

[23] H. Cai *et al.*, "Source authentication and path validation in networks using orthogonal sequences," in *2016 25th International Conference on Computer Communication and Networks (ICCCN)*, 2016, pp. 1–10.

[24] B. Wu *et al.*, "Enabling efficient source and path verification via probabilistic packet marking," in *2018 IEEE/ACM 26th International Symposium on Quality of Service (IWQoS)*, 2018, pp. 1–10.

[25] F. Brockners, S. Bhandari, and T. Mizrahi, "Data Fields for In Situ Operations, Administration, and Maintenance (IOAM)," RFC 9197, May 2022. [Online]. Available: https://www.rfc-editor.org/info/rfc9197

[26] C. Filsfils *et al.*, "The segment routing architecture," in *IEEE Global Communications Conference (GLOBECOM)*, 2015, pp. 1–6.

[27] 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.

[28] R. S. Guimarães *et al.*, "M-polka: Multipath polynomial key-based source routing for reliable communications," *IEEE Transactions on Network and Service Management*, 2022.

BIOGRAPHIES

Everson Scherrer Borges received his B.Sc. degree in Information of Systems from the Faculty of São Camilo-ES, Cachoeiro de Itapemirim, Espirito Santo, Brazil, in 2004, and his M.Sc. degree in operational research and computational intelligence from university Cândido Mendes, Campos Rio de Janeiro, Brazil, 2011. He started his Ph.D. in 2021, working at the "Núcleo de Estudos em Redes definidas por Software Lab", under the supervision of Prof. Magnos Martinello. He is an Associate Professor with the Department of Information Systems, Instituto Federal do Espirito Santo, Campus Cachoeiro de Itapemirim, Brazil. He was also worked as a CONNECT Research Assistant at Trinity College Dublin under the supervision of Prof. Marco Ruffini and was involved in FreeSpace projects. His research interests include network softwarization, network security, network function virtualization, and programmable data planes.

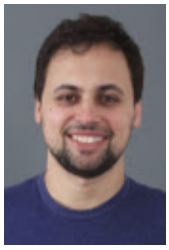
Magnos Martinello holds the position of Associate Professor within the Department of Informatics (DI) at the Federal University of Espírito Santo (UFES) in Brazil. He received his PhD in Computer Science from the Institut National Polytechnique de Toulouse (INPT) in 2005, developing his research at LAAS-CNRS. Recently, he served as visiting researcher at the University of Waikato (2023-2024). He has contributed to numerous journals and conferences and acting as TPC member and Program Chair in the field of computer networks. His current research interests include software-defined networks, and in-network artificial intelligence.

Vitor Bonella is a graduate of Computer Science from the Federal University of Espirito Santo, currently pursues a master's degree with a specialization in Artificial Intelligence. His academic endeavors have centered on research within the realms of AI and Networks, showcasing a commitment to advancing knowledge in these fields.

Abraão Santos is presently undertaking his undergraduate studies at the Federal University of Espírito Santo, specializing in Computer Science. His academic pursuits revolve around the realms of virtualization and system development, with a focus on devising solutions or applications tailored to address networking challenges.

Roberta L. Gomes Bachelor's in Computer Systems Engineering from the Federal University of Espírito Santo - Ufes (1999), Master's in Computer Science from the Federal University of Rio de Janeiro (2001), and Doctorate in Computer Systems from Université Paul Sabatier / LAAS, Toulouse-FR (2006). Currently, holds the position of Full Professor in the Computer Science Department at Ufes. Has expertise in Computer Science - Distributed Systems, with a focus on Distributed Systems Integration, Social Internet of Things (SIoT), Smart Cities, and Blockchains. Since 2010, coordinates the Introcomp Project, which offers free programming courses to state school students.

Cristina Dominicini received her PhD degree in Computer Science from Federal University of Espírito Santo (UFES, Brazil) in 2019, her M.Sc. degree in Electrical Engineering from University of São Paulo (USP, Brazil) in 2012, and her B.Sc. degree in Computer Engineering from Federal University of Espírito Santo (UFES, Brazil) in 2009. In 2014, she joined the Department of Informatics at Instituto Federal do Espírito Santo (IFES, Brazil) as Assistant Professor. Her research interests include Network Functions Virtualization, Software-Defined Networks and Next Generation Networks.



Rafael Silva Guimarães received his B.Sc. degree in Information of Systems from the Faculty of Vila Velha, Vila Velha, Brazil, in 2005, and his M.Sc. degree in Computing from the Federal University of Espírito Santo, Vitória, Brazil, in 2015. He is Ph.D. student in computer science at the Federal University of Espírito Santo. He is currently an associate professor at the Federal Institute of Espírito Santo and worked for 13 years with operation and advanced support using open source solutions. In his job, he was in charge to developing and managing the control of network using common protocols like BGP, DHCP, security policies etc. His research interests include software-defined networks and cross-layer orchestration. Also, his interest in self-organising networks models applied in a context of cross-layer orchestration.

Gabriel Tetzner Currently in his second year of studying Information Systems at the Alegre Campus of the Federal University of Espírito Santo (UFES). Concurrently, he is actively involved in the GT-OnE! project within the National Research and Educational Network (RNP), which specializes in monitoring optical fibers in cloud environments. His academic endeavors resonate closely with his interests, spanning Programming, Computer Networks, Information Security, and DevOps methodologies.

Marinho Barcellos is a Senior Lecturer at the University of Waikato, New Zealand. Before, he was an Associate Professor at UFRGS, Brazil. He has contributed with research outputs in areas including network security, Internet measurements, and programmable data planes. Barcellos has also contributed his expertise to the research community acting as Program Chair and TPC member of multiple conferences in these areas.

Marco Ruffini received his M.Eng. in telecommunications engineering in 2002 from Polytechnic University of Marche, Italy. After working as a research scientist for Philips in Germany, he joined Trinity College Dublin in 2005, where he received his Ph.D. in 2007. His main research is in the area of 5G optical networks, where he carries out pioneering work on the convergence of fixed-mobile and access-metro networks, and on the virtualisation of next generation networks, and has been invited to share his vision through several keynote and talks at major international conferences across the world and the OpenIreland beyond 5G testbed research infrastructure. He has recently started working also on quantum networking where he is collaborating with the US Centre for Quantum Networks (CQN). He authored over 200 international publications, 10 patents, contributed to industry standards and secured research funding for over € 14 milion, and contributed is novel virtual Dynamic Bandwidth Allocation (vDBA) concept to the BroadBand Forum standardisation body.