An Implementation of Verifiable Routing on PolKA

Henrique Coutinho Layber†, Roberta Lima Gomes†, Magnos Martinello†, Vitor B. Bonella†, Everson S. Borges‡, Rafael Guimarães†‡

†Department of Informatics, Federal University of Espírito Santo
‡Department of Informatics, Federal Institute of Education Science and Technology of Espírito Santo

Abstract

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magnam aliquam quaerat voluptatem. Ut enim aeque doleamus animo, cum corpore dolemus, fieri tamen permagna accessio potest, si aliquod aeternum et infinitum impendere malum nobis opinemur. Quod idem licet transferre in voluptatem, ut postea variari voluptas distinguique possit, augeri amplificarique non possit. At etiam Athenis, ut e patre audiebam facete et urbane Stoicos irridente, statua est in quo a nobis philosophia defensa et collaudata est, cum id, quod maxime placeat, facere possimus, omnis voluptas assumenda est, omnis dolor repellendus. Temporibus autem quibusdam et.

Keywords

Verifiable Routing; Path Verification; Proof-of-transit; In-networking Programming

1. Introduction

Ever since Source Routing (SR) was proposed, there has been a need to ensure that packets traverse the network along the paths selected by the source, not only for security reasons but also to ensure that the network is functioning correctly and correctly configured. This is particularly important in the context of Software-Defined Networking (SDN), where the control plane can select paths based on a variety of criteria.

In this paper, we propose a new P4[1] implementation for a new protocol layer for PolKA[2], able to do verify the actual route used for a packet. It is available on GitHub[1]. This is achieved by using a composition of hash functions on stateless core switches, each using a key to generate a digest that can be checked by the controller which knows the secrets. The controller can then verify that the packet traversed the network along the path selected by the source, ensuring that the network is functioning correctly.

2. Problem Definition

Let i be the source node (**i**ngress node) and e be the destination node (**e**gress node). Let path P be a sequence of nodes:

$$\underset{i \to e}{P} = (i, s_1, s_2, ..., s_{n-1}, s_n, e)$$

where
P Path from i to e.
s_n Core switch n in the path.

[1] https://github.com/Henriquelay/polka-halfsiphash/tree/remake/mininet/polka-example

n Number of core switches in the path.
i Ingress edge (source).
e Egress edge (destination).

In PolKA, the route up to the protocol boundary (usually, the SDN border) is defined in i [3]. i sets the packet header with enough information for each core node to calculate the next hop. Calculating each hop is done using Chinese Remainder Theorem (CRT) and the Residue Number System (RNS)[4], and is out of the scope of this paper. All paths are assumed to be both valid and all information correct unless stated otherwise.

The main problem we are trying to solve is path validation, that is, to have a way to ensure if the packets are actually following the path defined. Notably, it does not require verification, that is, listing the switches traversed is not required.

A solution should be able to identify if:
1. The packet has passed through the switches in the path.
2. The packet has passed through the correct order of switches.
3. The packet has not passed through any switch that is not in the Path.

More formally, given a sequence of switches $\underset{i \to e}{P}$, and a captured sequence of switches actually traversed P_j, a solution should identify if $\underset{i \to e}{P} = P_j$.

3. Solution Proposal

Each node's execution plan is stateless and can alter the header of the packet, which we will use to detect if the path taken is correct. So, a node s_i can be viewed as a function $g_{s_i}(x)$.

In order to represent all nodes by the same function (for implementation purposes), we assign a distinct value k for each s node, and use a bivariate function $f\left(k_{s_i}, x\right) = f_{s_i}(x)$. By using functions in two variables, we force one of the variables to have any uniquely per-node value, ensuring that the function is unique for each switch, that is, $f_{s_y}(x) \neq f_{s_z}(x) \Leftrightarrow y \neq z$.

Using function composition is a good way to propagate errors since it preserves the order-sensitive property of the path, since $f \circ g \neq g \circ f$ in a general case. Each node will execute a single function of this composition, using the previous node's output as input. In this way:

$$\left(f_{s_1} \circ f_{s_2} \circ f_{s_3}\right)(x) = f\left(k_{s_3}, f\left(k_{s_2}, f\left(k_{s_1}, x\right)\right)\right)$$

s_i i-th switch in the path.
$f_{s_i}(x)$ Function representing switch s_i.
k_{s_i} Unique identifier for switch s_i.

3.1. Assumption

3.2. Implementation

3.2.1. Setup

All implementation and experiments took place on a VM[2] setup with Mininet-wifi[5], and were targeting Mininet's[6] Behavioral Model version 2 (BMv2)

VM mininet-wifi BMv2 etc

[2]Available on PolKA's repository https://github.com/nerds-ufes/polka

3.2.2. Code

By making the function f is a checksum function, and the unique identifier k_{s_i} as the `switch_id`, we apply an input data into a chain checksum functions and verify if they match. The controller will act as a validator, since it already has access to all `switch_id`. For additional verification, we also integrate the calculated exit port into the checksum, covering some other forms of

It was implemented as a version on PolKA, this means it uses the same protocol identifier `0x1234` and is interoperable with PolKA.

Figure 1 shows the used topology used in the experiments.

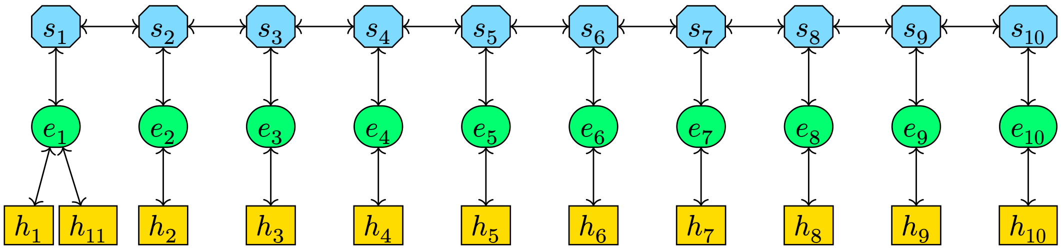

Figure 1: Topology setup.
s_n are core switches, e_n are edge switches, h_n are hosts.

It was validated with {{THIS}}.

This detects {{X Y Z}}

4. Limitations

- Replay attack is undetectable if timing is not considered.

5. Future Work

- Rotating key for switches for detecting replay attacks (holy shit this is hard)
- Include entrance port in checksum

Bibliography

[1] P. Bosshart *et al.*, "P4: programming protocol-independent packet processors," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 3, pp. 87–95, Jul. 2014, doi: 10.1145/2656877.2656890.

[2] 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. doi: 10.1109/NetSoft48620.2020.9165501.

[3] E. S. Borges *et al.*, "PoT-PolKA: Let the Edge Control the Proof-of-Transit in Path-Aware Networks," *IEEE Transactions on Network and Service Management*, vol. 21, no. 4, pp. 3681–3691, 2024, doi: 10.1109/TNSM.2024.3389457.

[4] 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. doi: 10.23919/ONDM51796.2021.9492363.

[5] R. Fontes, S. Afzal, S. Brito, M. Santos, and C. Esteve Rothenberg, "Mininet-WiFi: Emulating Software-Defined Wireless Networks," in *2nd International Workshop on Management of SDN and NFV Systems, 2015(ManSDN/NFV 2015)*, Nov. 2015.

[6] B. Lantz, B. Heller, and N. McKeown, “A network in a laptop: rapid prototyping for software-defined networks,” in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, in Hotnets-IX. Monterey, California: Association for Computing Machinery, 2010. doi: 10.1145/1868447.1868466.