PolKA: Polynomial Key-based Architecture for Source Routing

Cristina K. Dominicini\textsuperscript{1}, Rafael Silva Guimarães\textsuperscript{1}, Magnos Martinello\textsuperscript{2}, Moises R. N. Ribeiro\textsuperscript{2}, Rodolfo Villaça\textsuperscript{2}, Diego Mafioletti\textsuperscript{1}, Ana Locatelli\textsuperscript{2}, Everson Borges\textsuperscript{2}, Edgard Cunha\textsuperscript{2}, and Isis Oliveira\textsuperscript{1}

\textsuperscript{1}Federal Institute of Espírito Santo, \textsuperscript{2}Federal University of Espírito Santo, Contact: rafaelg@ifes.edu.br
Which PANRG's problem does PolKA solve?

“(...) Endpoints have very little information about the paths over which their traffic is carried, **and no control at all beyond the destination address**. (...)”
Can SDN table-based solutions offer path-aware control?

- Problems:
  - Large number of states → **Scalability 😞**
  - Limited capacity of tables → **Granularity 😞**
  - Latency for path configuration → **Agility 😞**

("(...) Endpoints have very little information about the paths over which their traffic is carried, **and no control at all beyond the destination address. (...)”

---

**SDN Table-based**

Reconfigurations for path migration
PANRG: The Control Problem

- “(...) Endpoints have very little information about the paths over which their traffic is carried, and no control at all beyond the destination address. (...)”

Can SDN table-based solutions offer path-aware control?

- Problems:
  - Large number of states → **Scalability 😞**
  - Limited capacity of tables → **Granularity 😞**
  - Latency for path configuration → **Agility 😞**
Source Routing: a key mechanism for endpoints

- **Endpoints controlling paths**: setup routeID at the edges
Why use PolKA as Strict Source Routing?

- Only PolKA Source Routing simultaneously meets the following requirements:

  - topology agnostic
  - no tables in the core
  - implementable in prog. switches
  - singlepath & multipath

  - general solution for path-aware networking
  - fixed header
  - exploit wire-speed and low latency dataplanes
  - path expressiveness reliability by exploring multiple paths
  - efficient in-band telemetry
  - fast path reconfiguration
  - no var header limitation with respect to hop limit
How does PolKA work?

- Three polynomials:
  - `routeID`: a route identifier calculated using the CRT (Chinese Remainder theorem).
  - `nodeID`: to identify each core node.
    - Irreducible polynomial
  - `portID`: to identify the ports of each core node.

- The forwarding uses a `mod` operation (remainder of division):
  
  \[
  \text{portID} = \langle \text{routeID} \rangle_{\text{nodeID}}
  \]
Timeline

PolKA paper IEEE NetSoft
- Novel Polynomial RNS-based SR and reuse of CRC hardware
- Emulated prototype in Mininet & Hardware prototype in SmartNICs

M-PolKA received the Intel Connectivity Research Grant (Fast Forward Initiative)

2020
- PolKA paper IEEE ONDM paper
- Deployment @ Pan-European GÉANT Testbed RARE
- PolKA data plane implementation in intercontinental testbed
- Hardware: Intel Tofino

2021/1
- IEEE ONDM paper

2021/2
- Integration with RARE+FreeRouter
- Emulated prototype in FreeRouter & Hardware prototype in Intel Tofino with FreeRouter control plane

2022
- M-PolKA paper IEEE TNSM (accepted for publication)
- Innovative apps: inband network telemetry, and load balance
- Emulated prototype in FreeRouter & Hardware prototype in Intel Tofino with FreeRouter control plane
- Extension to multipath SR for reliable communications
How to implement PolKA?

- **Reuse CRC hardware** to offer polynomial mod.
  - Externs in P4 language.
  - Support in high-performance Tofino switches.

- **RARE**: Open source full-featured router on networking hardware for R&E
  - **data plane**: P4 (bmv2 and Tofino) and DPDK
  - **control plane**: FreeRouter
    - Reuse of standard distributed protocols
    - Static table maps Segment Routing indexes to nodeIDs
    - Get available topology info from link-state protocols
How does PolKA work?

- The **Controller** chooses a path for a specific flow:
  - A set of switches: \{0011,0111,1011\}
  - and their output ports: \{1 , 10, 110\}

\[
\begin{align*}
    s_1(t) &= t + 1 = 11 \\
    s_2(t) &= t^2 + t + 1 = 111 \\
    s_3(t) &= t^3 + t + 1 = 1011
\end{align*}
\]

\[
\begin{align*}
    o_1(t) &= 1 \\
    o_2(t) &= t = 10 \\
    o_3(t) &= t^2 + t = 110
\end{align*}
\]
How does PolKA work?

- When packets arrive, an action at ingress embeds routeID into the packets.

RouteID = 10000
How does PolKA work?

- Forwarding using \( \text{mod} \) operation: \( <10000>_{0011} = 1 \rightarrow \text{output port} \)
- No \textit{routeID} rewrite! No tables!
How does PolKA work?

- **Forwarding using** \textbf{mod} \textit{operation}: \( <10000> \mod 0111 = 10 \rightarrow \text{output port} \)
- **No routeID rewrite! No tables!**
How does PolKA work?

- Forwarding using `mod` operation: \(<10000>_{1011} = 110 \rightarrow \text{output port}\)
- No `routeID` rewrite! No tables!
How does PolKA work?

- Finally, an action at edge egress node removes `routeID`.
How does PolKA work?

- Packet is delivered to the application in a transparent manner.
Thank you!

Rafael Silva Guimaraes
rafaelq@ifes.edu.br

* This work was a recipient of the 2021 Google Research Scholar and the 2022 Intel Connectivity Research Grant (Fast Forward Initiative) Awards, and received funds from CAPES (Finance Code 001), CNPq, FAPESP, FAPES, CTIC, and RNP.