M-PolKA: Enabling and Exploiting Multipath Stateless Source Routing for Programmable Data Planes

Rafael Silva Guimarães[1], Cristina Klippel Dominicini[1], *Everson Scherrer Borges, Rodolfo Villaça*[2], *Magnos Martinello*[2], *Moises R. N. Ribeiro*[2],

[1]*Federal Institute of Espírito Santo,* [2]*Federal University of Espírito Santo,*

Contact: rafaelg@ifes.edu.br

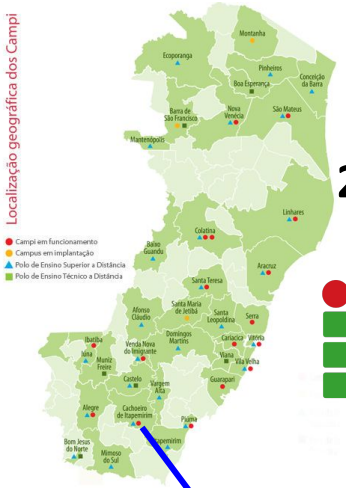
INSTITUTO FEDERAL
Espírito Santo

UFES

Location and Institutions

- **Espírito Santo, Brazil**

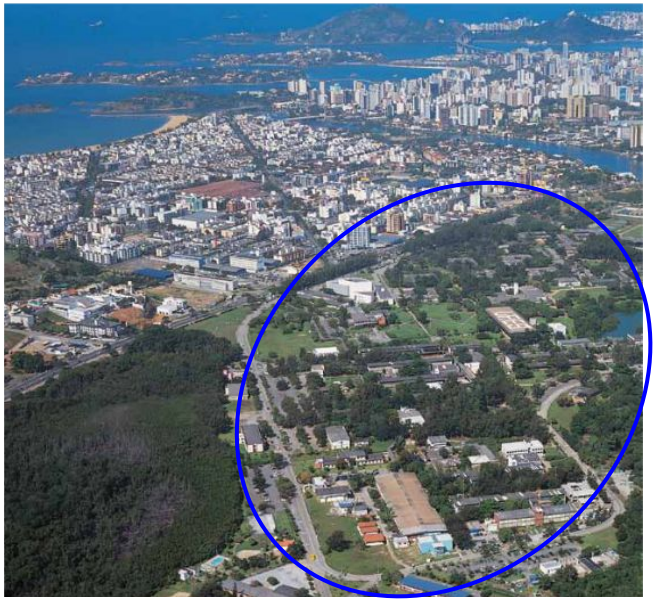
- **IFES: Assistant Professor**



22 campi

INSTITUTO FEDERAL
Espírito Santo

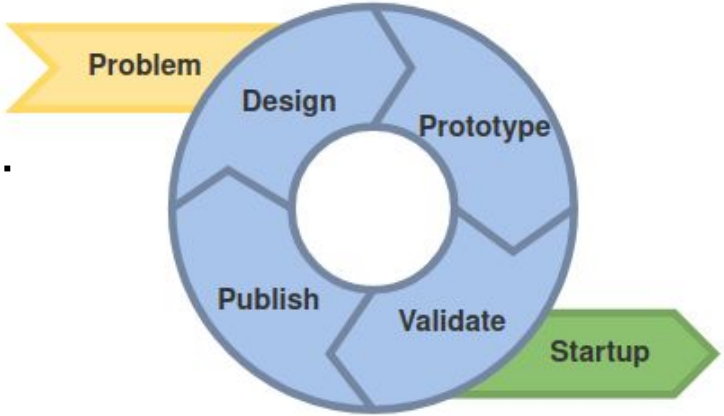
- **UFES: Collaborator**



LabNERDS: Software Defined Networks Research Group

- **Mission:** Innovate in networking systems
- **Areas**: SDN, NFV, autonomous networks, …

http://nerds.inf.ufes.br



Agenda

- **Motivation**
- Proposal
- Design
- Prototype
- Conclusions
- Ongoing applications

Motivation

- **SDN and Programmable Network Devices**:
  - Innovation and custom protocols.

Motivation

- **SDN and Programmable Network Devices**:
  - Innovation and custom protocols.
- **Bottleneck**: routing and forwarding based on **table entries**
  - Large number of states in the Network⟶ Low Scalability Control Planes
  - Limited TCAMs ⟶ Coarse Granularity for Traffic Engineering
  - High Latency for path configuration ⟶ No Agility in Traffic Engineering

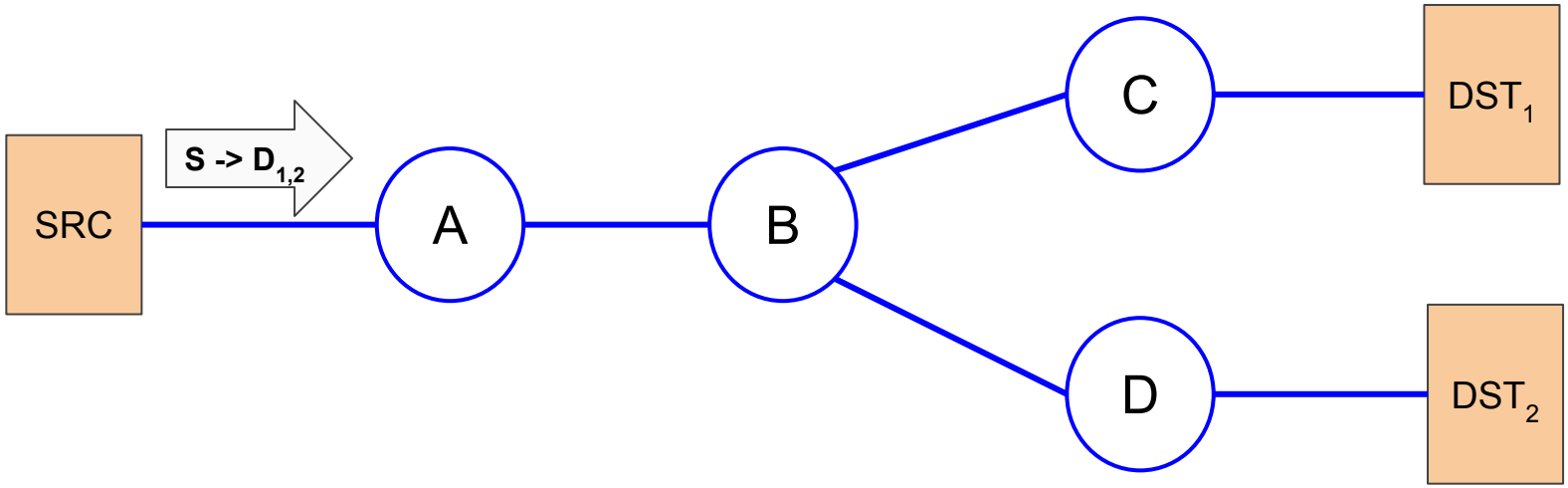
Motivation

- **SDN and Programmable Network Devices**:
  - Innovation and custom protocols.
- **Bottleneck**: routing and forwarding based on **table entries**
  - Large number of states in the Network⟶ Low Scalability Control Planes
  - Limited TCAMs ⟶ Coarse Granularity for Traffic Engineering
  - High Latency for path configuration ⟶ No Agility in Traffic Engineering
- What are the alternatives to tackle these problems?
  - **Source Routing (SR):**

    - A source specifies a path and adds a route label to the packet header.

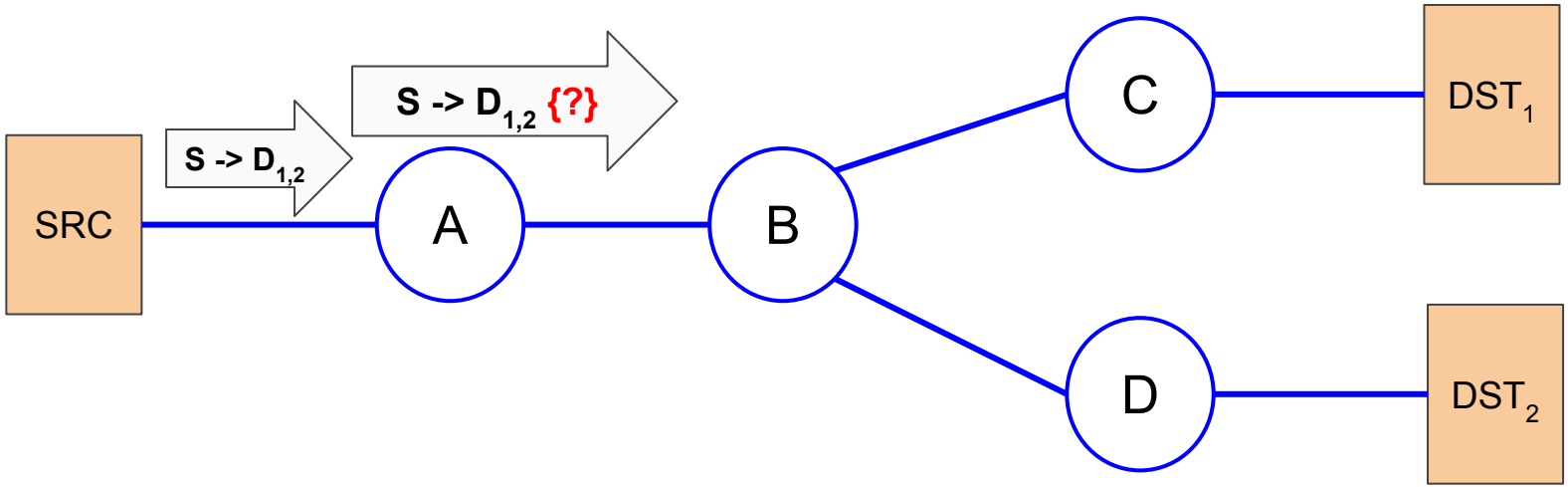
Motivation

- **Traditional way of representing List-based SR (LSR)**
  - The path is a list of ports or addresses.
  - Each node performs a pop operation.



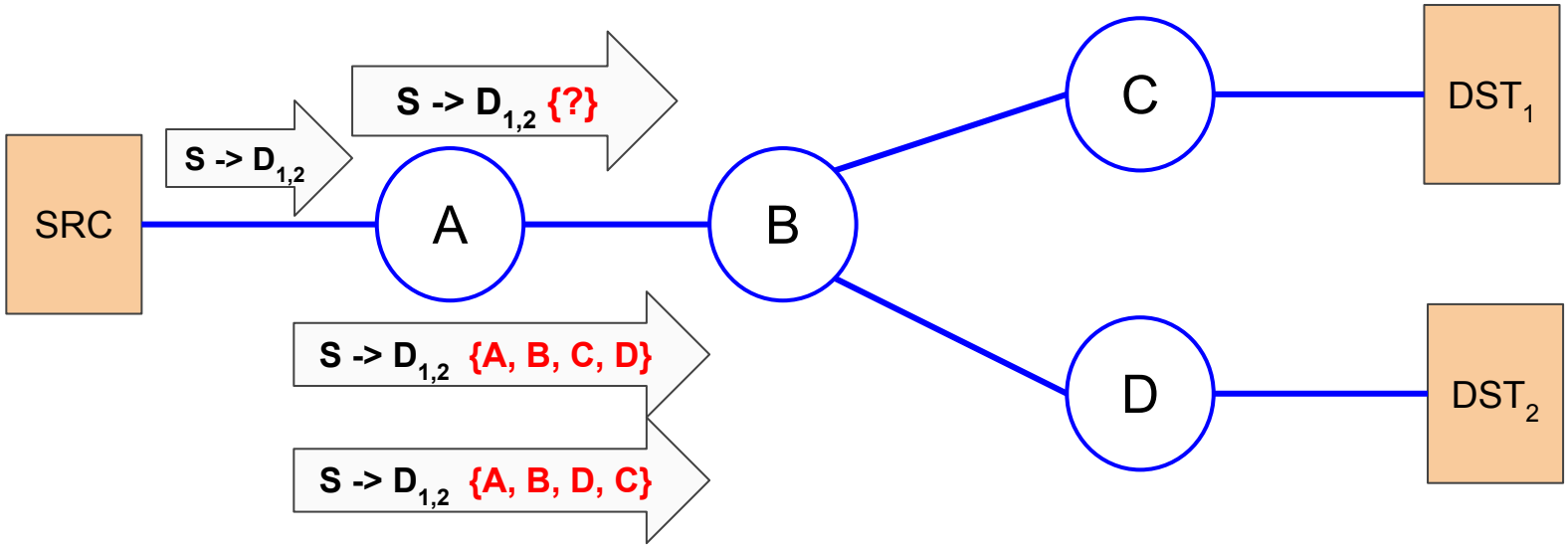
Motivation

- **Traditional way of representing List-based SR (LSR)**
  - The Source sends a Multicast packet to DST_1 and DST_2.
  - Router A determines that C and D are the Egress routers.



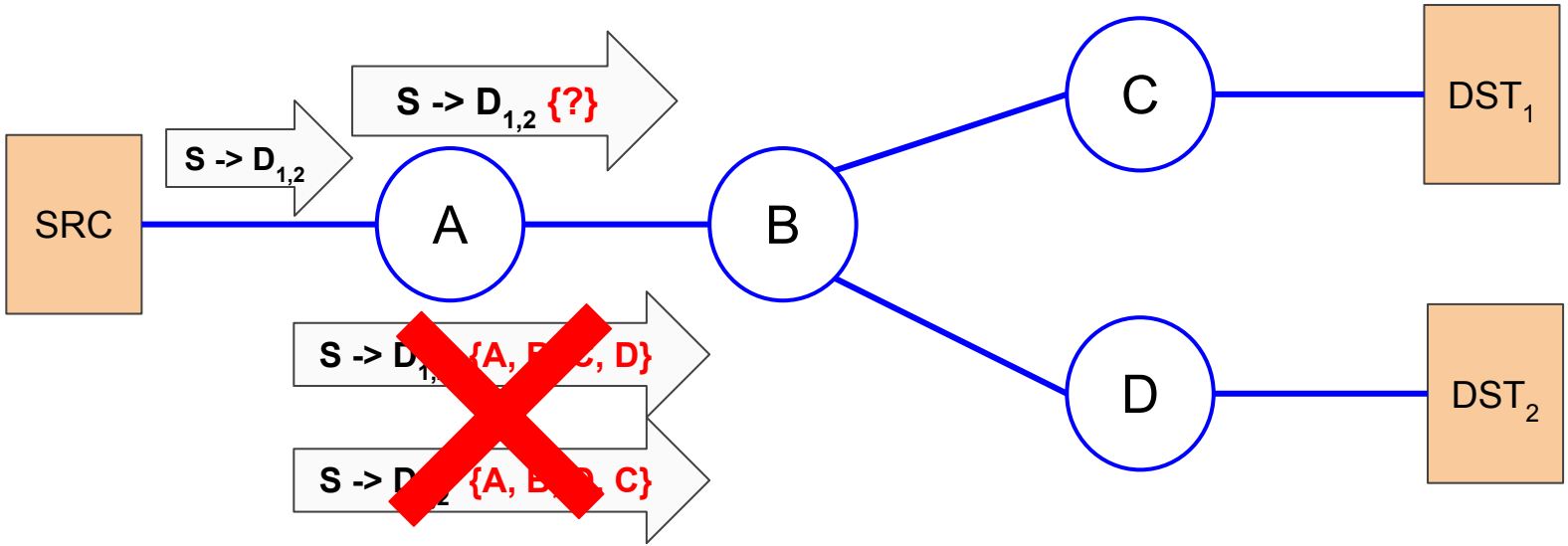
Motivation

- **Traditional way of representing List-based SR (LSR)**
  - Router B needs to replicate the packet to C and D



Motivation

- **Traditional way of representing List-based SR (LSR)**
  - SR needs a list of hops that does not represent a replication.



Agenda

- Motivation
- **Proposal**
- Design
- Prototype
- Conclusions
- Ongoing applications

M-PolKA Proposal

- To design a SR approach that simultaneously meets the requirements:

topology agnostic

no tables in the core

no state in the packet

implementable in prog. switches

encoded path

fixed header

multipath expressiveness

M-PolKA Proposal

- To design a SR approach that simultaneously meets the requirements:

topology agnostic

no tables in the core

no state in the packet

implementable in prog. switches

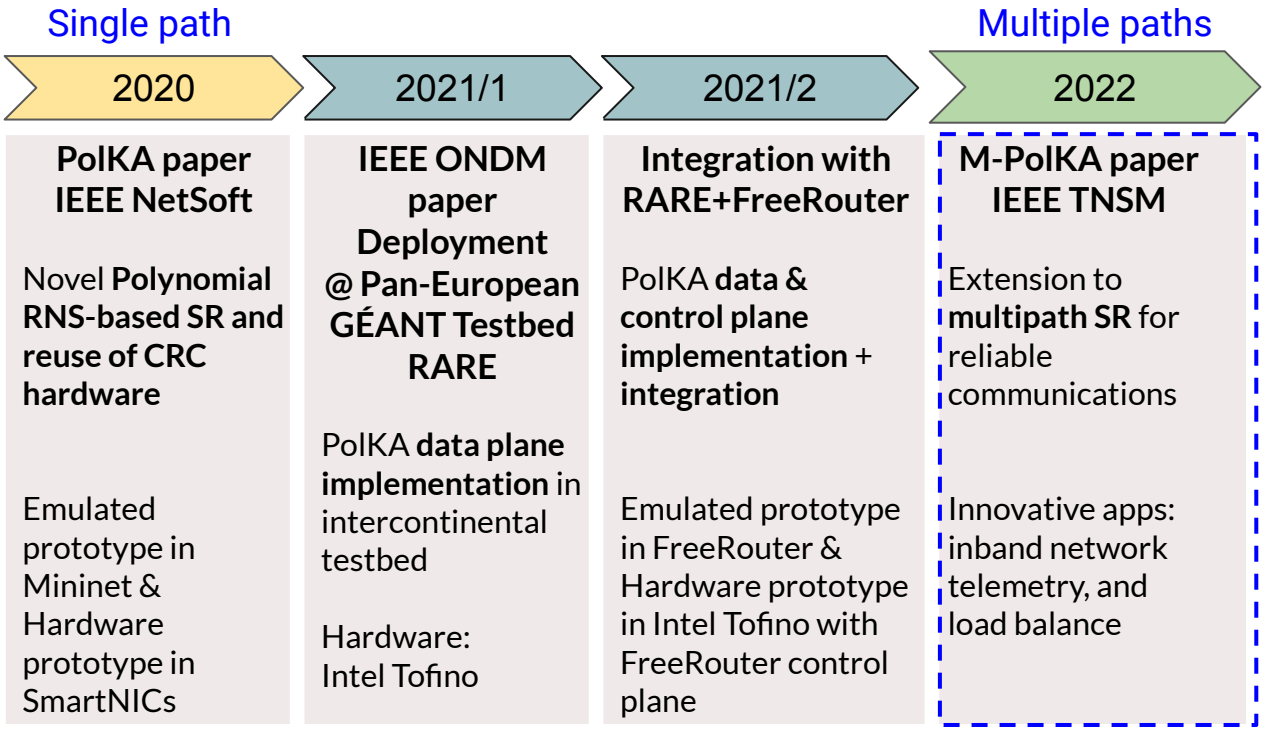
encoded path

fixed header

multipath expressiveness

- M-PolKA: Multipath Polynomial Key-based Architecture for Source Routing
  - Polynomial Residue Number System (**RNS**) (*Shoup, 2008*)
  - Chinese Remainder Theorem (**CRT**)
  - Forwarding based on an arithmetic operation: **remainder of division**
    - Path is encoded in a route label.
    - Each node decodes only its next hop with its own key.

Timeline


Single path
Multiple paths
2020
2021/1
2021/2
2022
PolKA paper IEEE NetSoft
Novel Polynomial RNS-based SR and reuse of CRC hardware
Emulated prototype in Mininet & Hardware prototype in SmartNICs
IEEE ONDM paper Deployment @ Pan-European GÉANT Testbed RARE
PolKA data plane implementation in intercontinental testbed
Hardware: Intel Tofino
Integration with RARE+FreeRouter
PolKA data & control plane implementation + integration
Emulated prototype in FreeRouter & Hardware prototype in Intel Tofino with FreeRouter control plane
M-PolKA paper IEEE TNSM
Extension to multipath SR for reliable communications
Innovative apps: inband network telemetry, and load balance

Timeline

INSTITUTO FEDERAL
Espírito Santo

PolKA received the 2021 Google Research Scholar Award

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

2020	2021/1	2021/2	2022
PolKA paper IEEE NetSoft Novel **Polynomial RNS-based SR and reuse of CRC hardware** Emulated prototype in Mininet & Hardware prototype in SmartNICs	**IEEE ONDM paper Deployment @ Pan-European GÉANT Testbed RARE** PolKA **data plane implementation** in intercontinental testbed Hardware: Intel Tofino	**Integration with RARE+FreeRouter** PolKA **data & control plane implementation** + **integration** Emulated prototype in FreeRouter & Hardware prototype in Intel Tofino with FreeRouter control plane	**M-PolKA paper IEEE TNSM** Extension to **multipath SR** for reliable communications Innovative apps: inband network telemetry, and load balance

Agenda

- Motivation
- Proposal
- **Design**
- Prototype
- Conclusions
- Ongoing applications

How does M-PolKA work?

- Three polynomials:
  - **routeID**: a route identifier calculated using the CRT.
  - **nodeID**: to identify each core node.

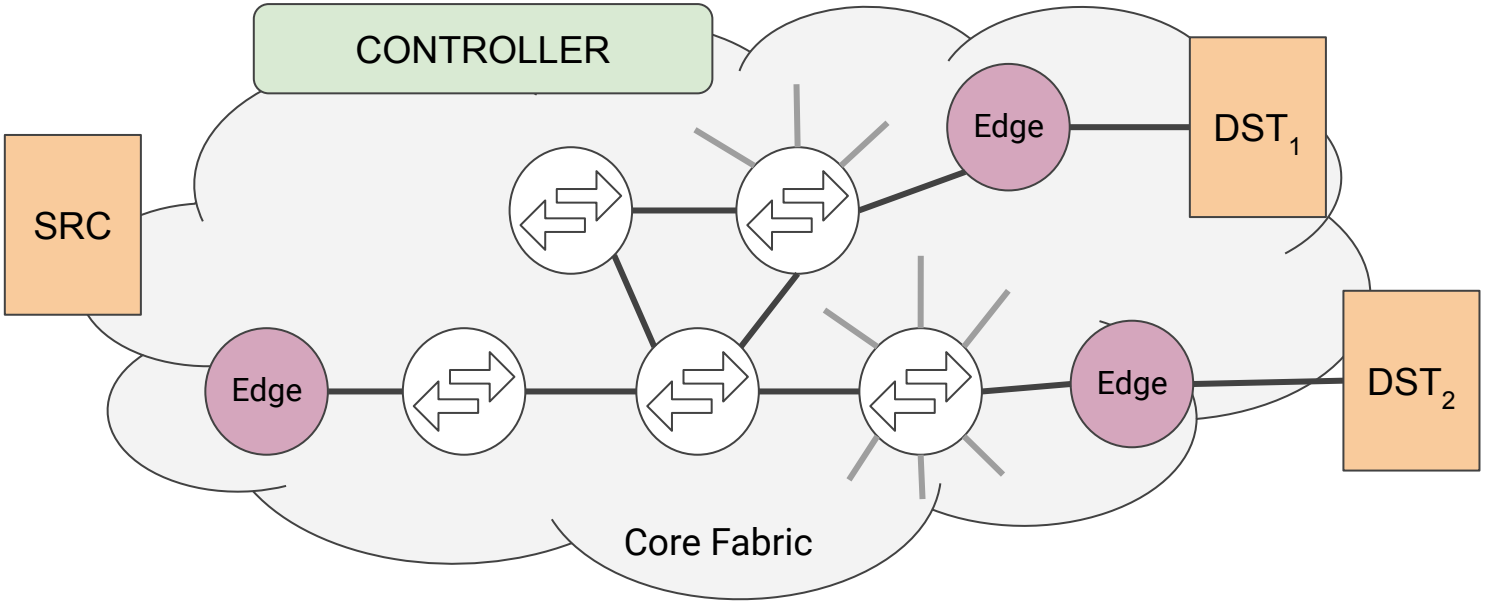
    - Irreducible polynomial
  - **tState**: to identify the transmission state on each node.

- The forwarding uses a **mod** operation (remainder of division):

$$tState = < routeID >_{nodeID}$$

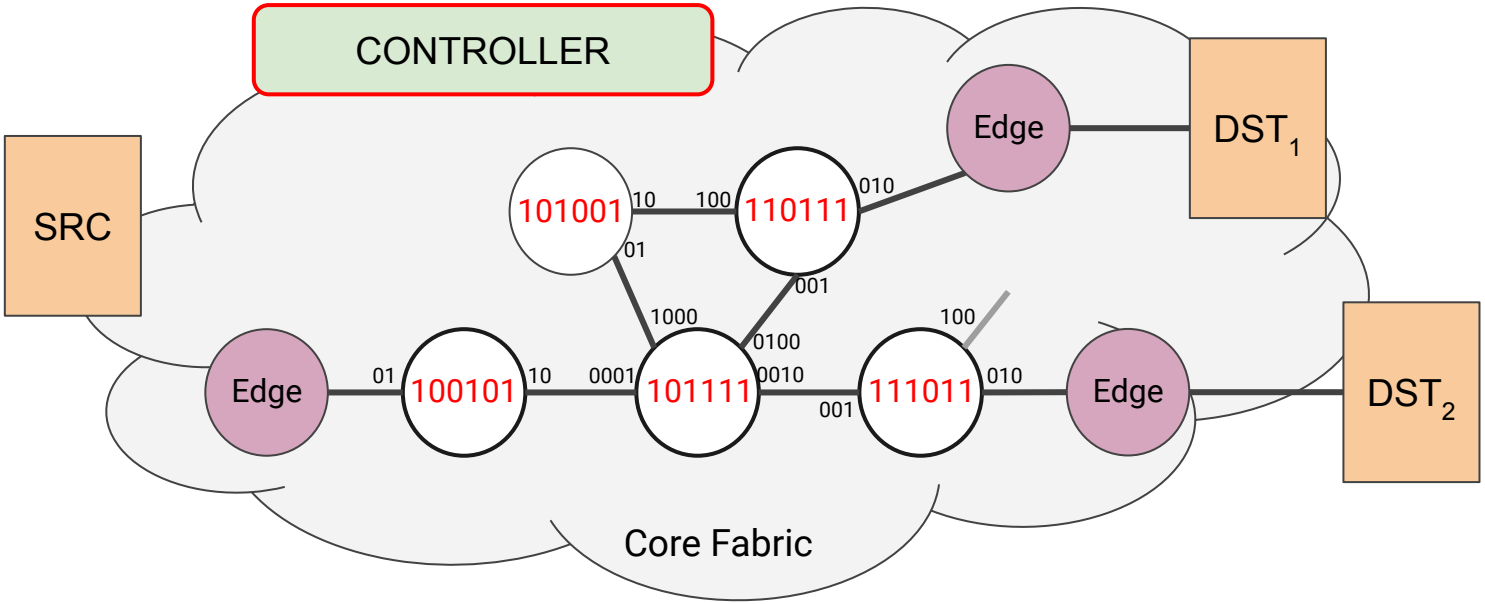
How does M-PolKA work?

- Hosts are connected to **edge switches.**
- Edges are connected to a fabric of **core switches.**



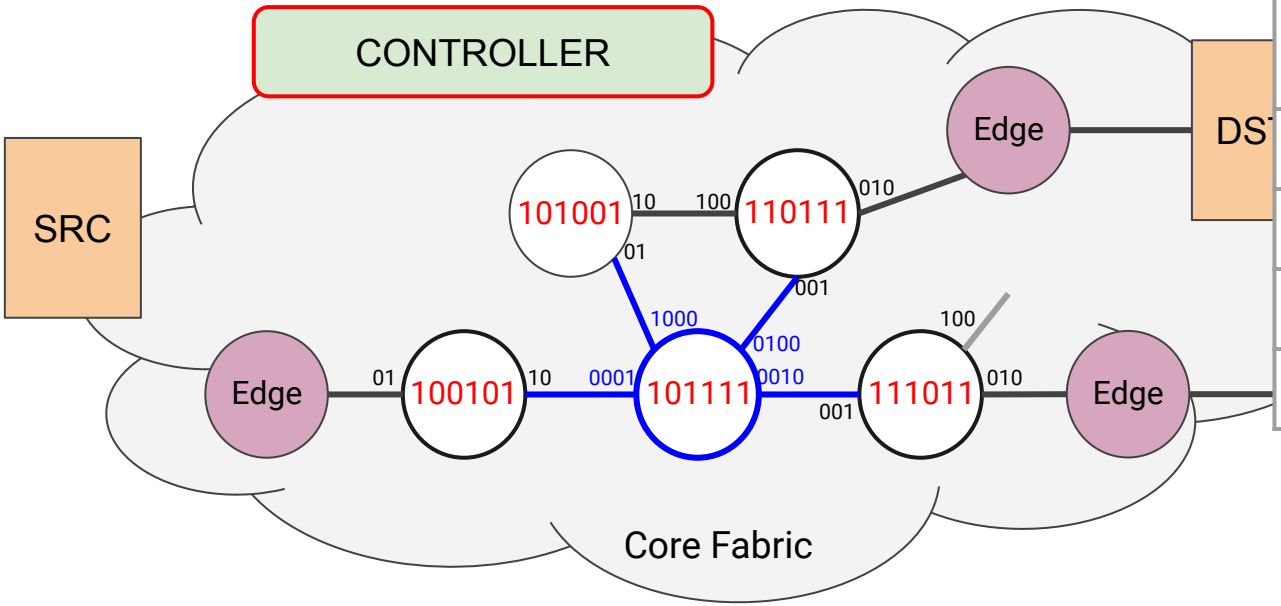
How does M-PolKA work?

- In a network configuration phase, the **Controller** assigns irreducible polynomials to core switches (***nodeIDs***).



How does M-PolKA work?

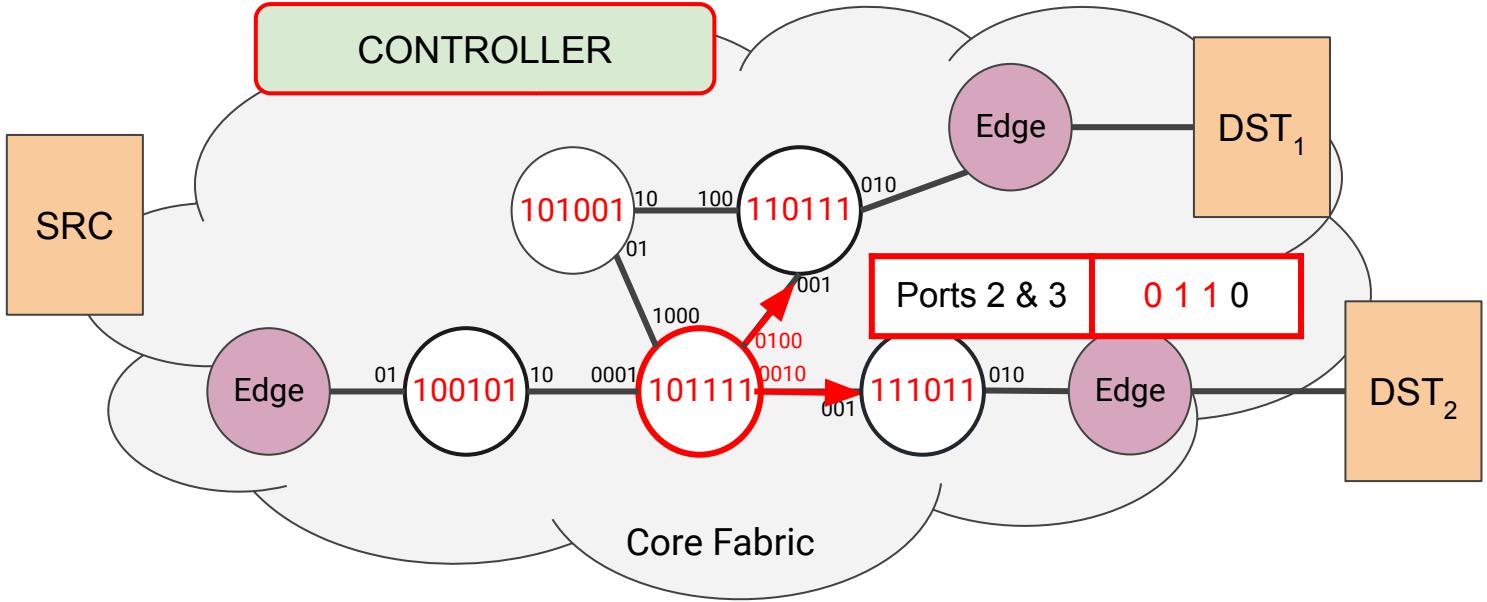
- In a network configuration phase, the **Controller** assigns irreducible polynomials to core switches (***nodeIDs***).
- Port labels are represented as binary polynomials marked by its position at the transmission state.



Port Number	TS Position
1	0001
2	0010
3	0100
4	1000

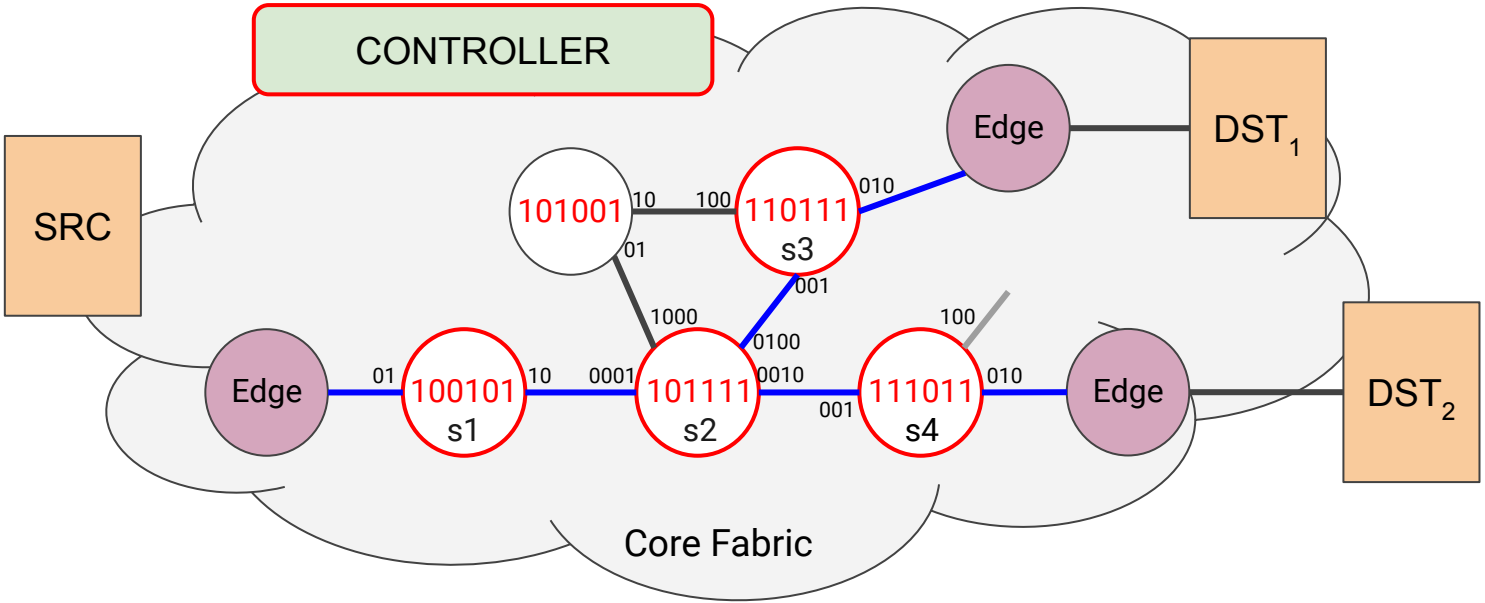
How does M-PolKA work?

- In a network configuration phase, the **Controller** assigns irreducible polynomials to core switches (***nodeIDs***).
- Port labels are represented as binary polynomials marked by its position at the transmission state list.



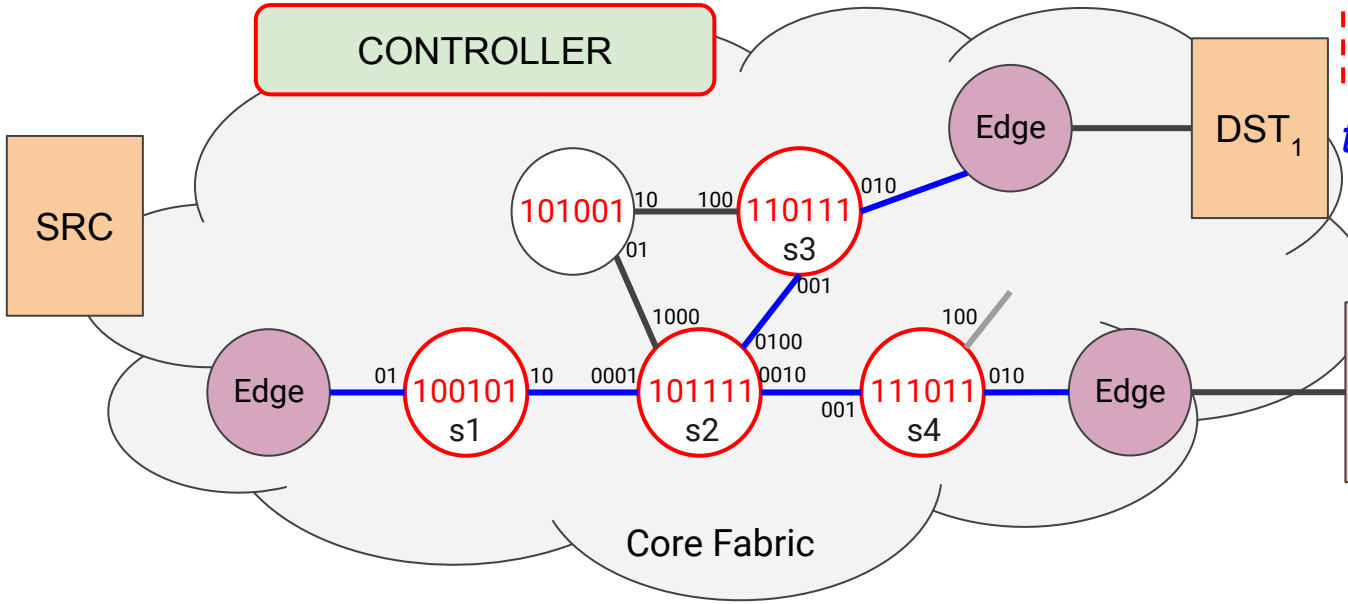
How does M-PolKA work?

- The **Controller** chooses a **path** for a specific flow (proactively or reactively):
  - A set of switches: {100101,101111,110111,111011}
  - and their output ports: {10 , 0110, 010,010}



How does M-PolKA work?

- The **Controller** chooses a **path** for a specific flow (proactively or reactively):
  - A set of switches: {100101,101111,110111,111011}
  - and their output ports: {10 , 0110, 010,010}



nodeID polynomials

$s_1(t) = t^5 + t^2 + 1$
$s_2(t) = t^5 + t^3 + t2 + t + 1$
$s_3(t) = t^5 + t^4 + t^2 + t + 1$
$s_4(t) = t^5 + t^4 + t^3 + t + 1$

tState polynomials

$o_1(t) = t = 10$
$o_2(t) = t2 + t = 110$
$o_3(t) = t = 010$
$o_4(t) = t = 010$

How does M-PolKA work?

- The **Controller** calculates the ***routeID*** using CRT:
  - Complexity: $\mathcal{O}(len(M)^2)$, where $M(t) = \prod_{i=1}^{N} s_i(t)$

R = 11110000101110101110

routeID

nodeID polynomials

$s_1(t) = t^5 + t^2 + 1$
$s_2(t) = t^5 + t^3 + t2 + t + 1$
$s_3(t) = t^5 + t^4 + t^2 + t + 1$
$s_4(t) = t^5 + t^4 + t^3 + t + 1$

tState polynomials

$o_1(t) = t = 10$
$o_2(t) = t2 + t = 110$
$o_3(t) = t = 010$
$o_4(t) = t = 010$

How does M-PolKA work?

- The **Controller** calculates the ***routeID*** using CRT:
  - Complexity: $\mathcal{O}(len(M)^2)$, where $M(t) = \prod_{i=1}^{N} s_i(t)$

R = 11110000101110101110

routeID

- Forwarding:

tState = < routeID >$_{nodeID}$

10	=	<11110000101110101110>$_{100101}$
110	=	<11110000101110101110>$_{101111}$
010	=	<11110000101110101110>$_{110111}$
010	=	<11110000101110101110>$_{111011}$

nodeID polynomials

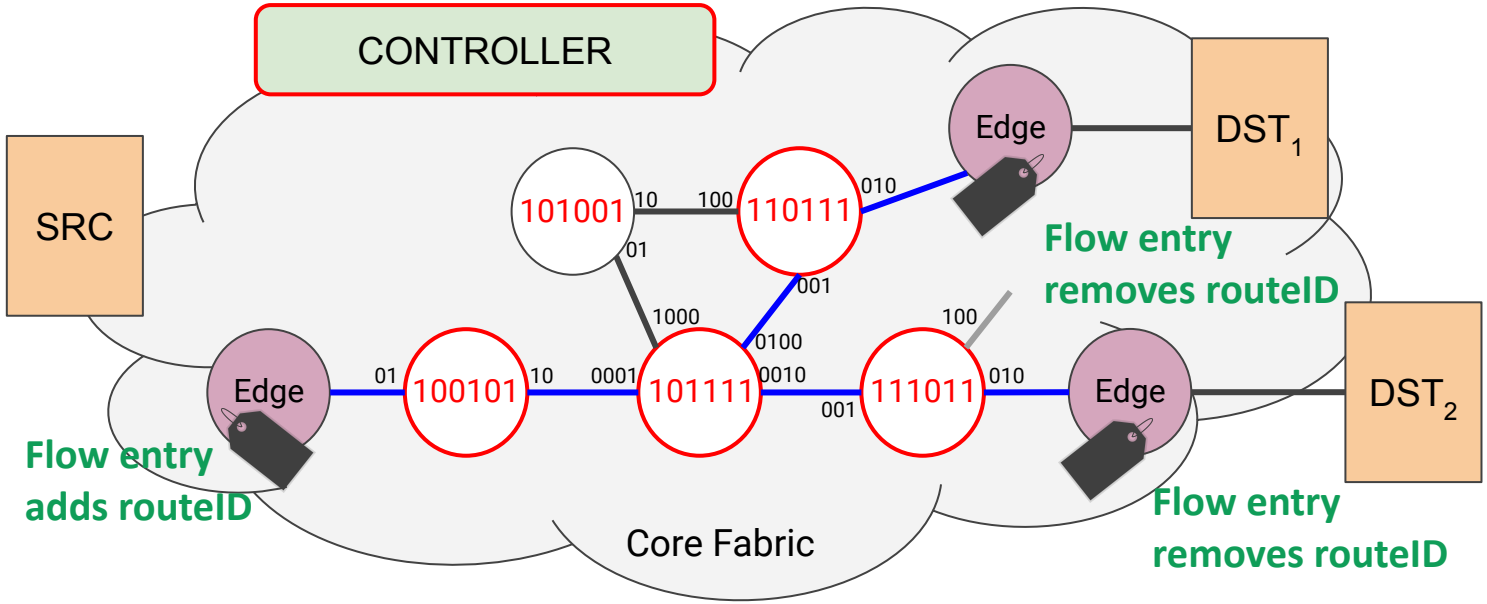
$s_1(t) = t^5 + t^2 + 1$
$s_2(t) = t^5 + t^3 + t2 + t + 1$
$s_3(t) = t^5 + t^4 + t^2 + t + 1$
$s_4(t) = t^5 + t^4 + t^3 + t + 1$

tState polynomials

$o_1(t) = t = 10$
$o_2(t) = t2 + t = 110$
$o_3(t) = t = 010$
$o_4(t) = t = 010$

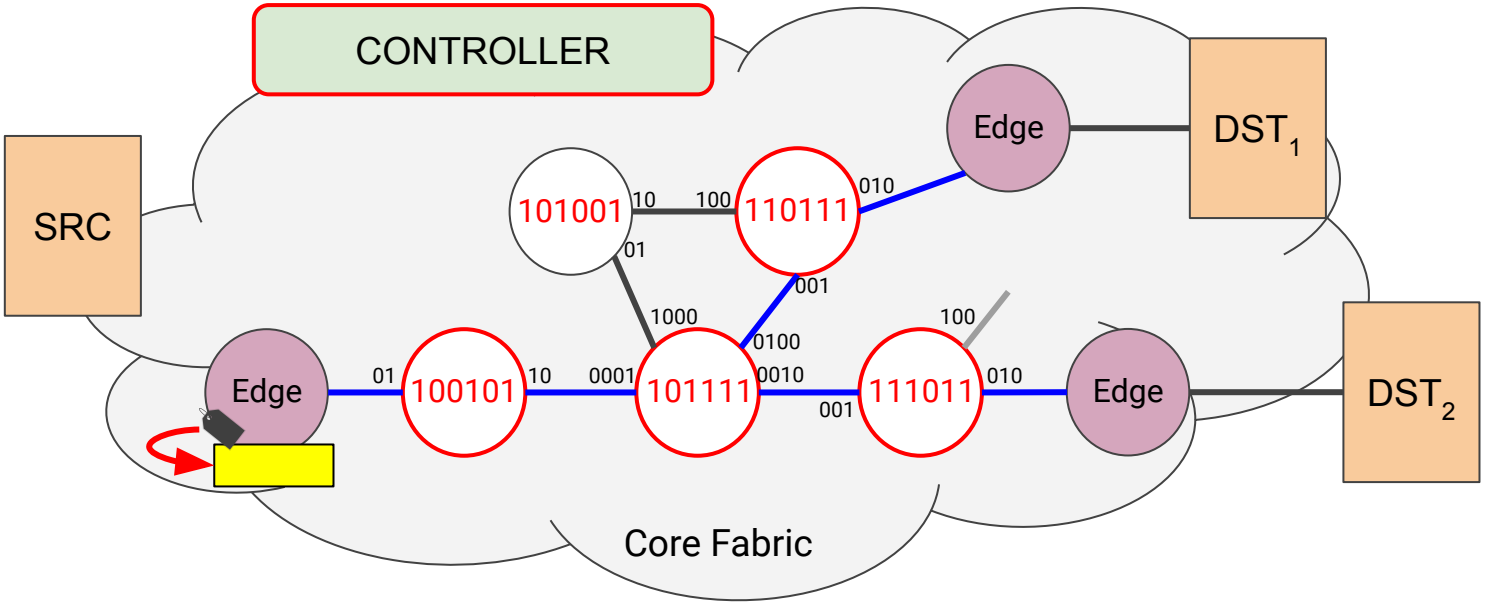
How does M-PolKA work?

- The **Controller** installs **flow entries** at the edges to add/remove ***routeIDs***.



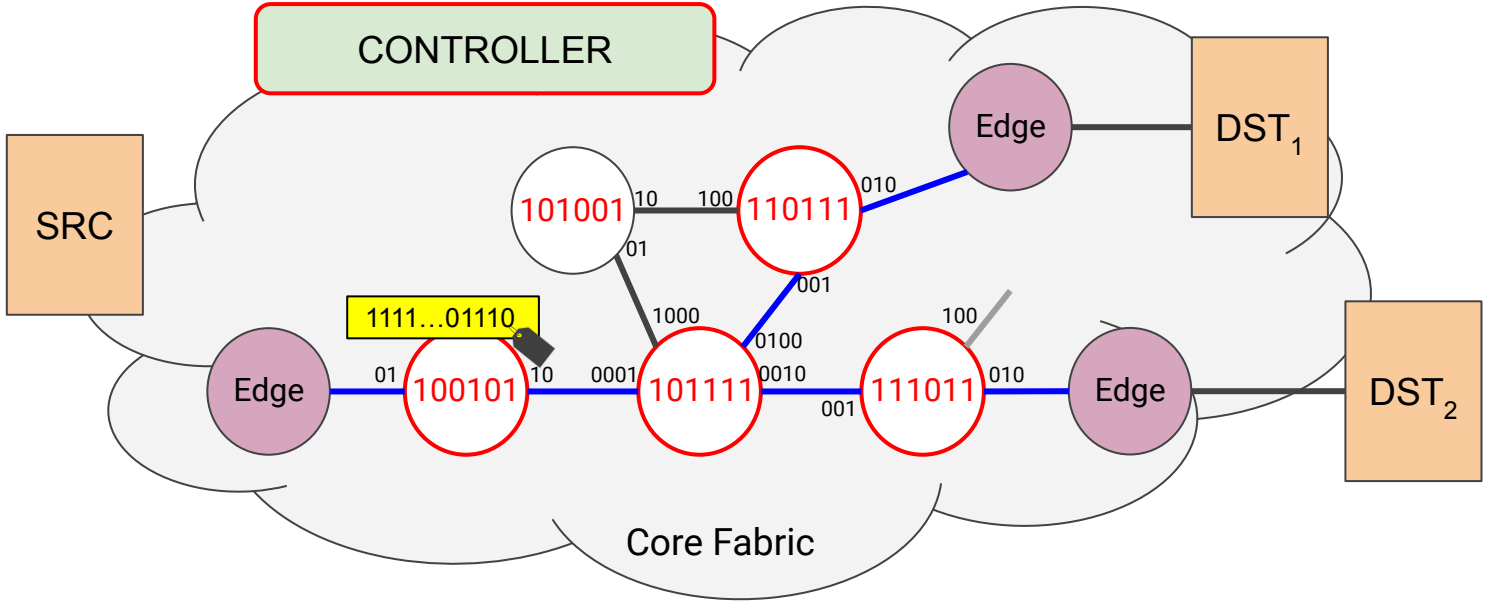
How does M-PolKA work?

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



How does M-PolKA work?

- Forwarding using **mod** operation:
  - <111100001011101011 10>$_{100101}$ = 10 → steer to the port 2
- No *routeID* rewrite! No tables!



How has the CRC Custom been implemented in P4?

- **P4 language does not natively support the mod operation.**
- **CRC hardware** (Cyclic Redundancy Check) offers polynomial mod.
  - The Tofino Native Architecture (**TNA**) supports **custom** CRC polynomials.
  - MOD = 2 SHIFTs + 1 CRC + 2 XORs

set_crc16_parameters calc 0x002b 0x0 0x0 false false

```
hash(
        nresult,
        HashAlgorithm.crc16_custom,
        nbase,
        {ndata},ncount
);
```

BMv2

set irreducible polynomial via gRPC interface

```
CRCPolynomial<bit<16>>(
                        coeff    = (65539 & 0xffff),
                        reversed = false,
                        msb      = false,
                        extended = false,
                        init     = 16w0x0000,
                        xor      = 16w0x0000) poly;
Hash<bit<16>>(HashAlgorithm_t.CUSTOM, poly) hash;
```

TOFINO

How has the CRC Custom been implemented in P4?

- **P4 language does not natively support the mod operation.**
- **CRC hardware** (Cyclic Redundancy Check) offers polynomial mod.
  - The Tofino Native Architecture (**TNA**) supports **custom** CRC polynomials.
  - MOD = 2 SHIFTs + 1 CRC + 2 XORs

command line only

```
set_crc16_parameters calc 0x002b 0x0 0x0 false false
```

```
hash(
        nresult,
        HashAlgorithm.crc16_custom,
        nbase,
        {ndata},ncount
);
```

BMv2

set irreducible polynomial via gRPC interface

```
CRCPolynomial<bit<16>>(
                        coeff    = (65539 & 0xffff),
                        reversed = false,
                        msb      = false,
                        extended = false,
                        init     = 16w0x0000,
                        xor      = 16w0x0000) poly;
Hash<bit<16>>(HashAlgorithm_t.CUSTOM, poly) hash;
```

TOFINO

How has the CRC Custom been implemented in P4?

- **P4 language does not natively support the mod operation.**
- **CRC hardware** (Cyclic Redundancy Check) offers polynomial mod.
  - The Tofino Native Architecture (**TNA**) supports **custom** CRC polynomials.
  - MOD = 2 SHIFTs + 1 CRC + 2 XORs

set_crc16_parameters calc 0x002b 0x0 0x0 false false

```
hash(
        nresult,
        HashAlgorithm.crc16_custom,
        nbase,
        {ndata},ncount
);
```

BMv2

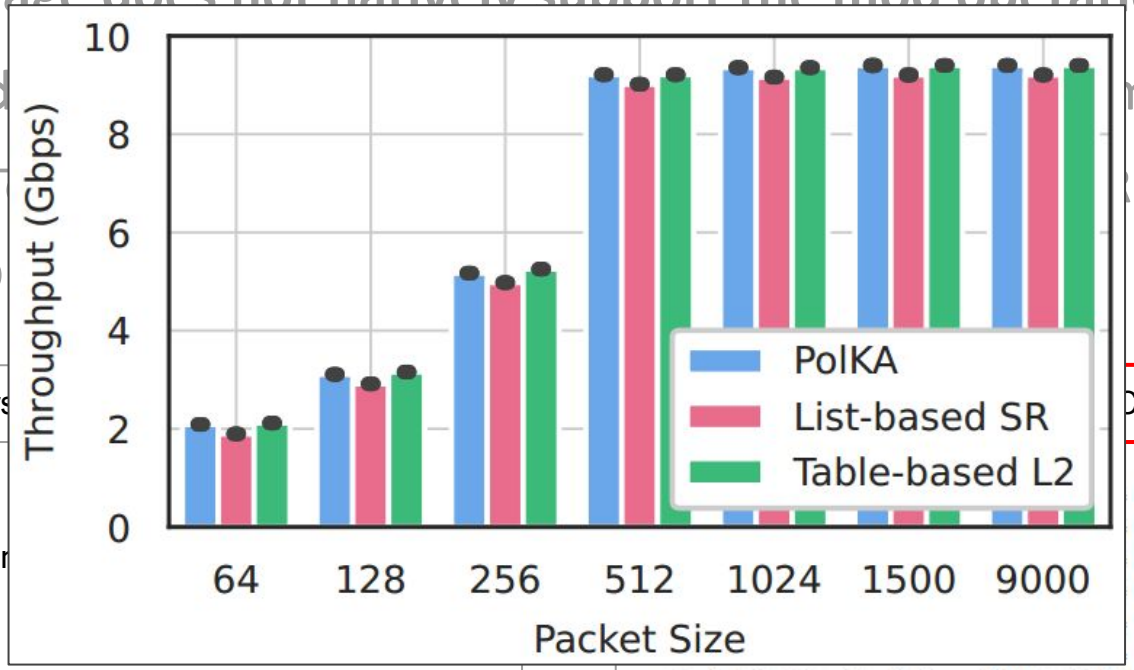
set irreducible polynomial via gRPC interface

```
CRCPolynomial<bit<16>>(
                        coeff    = (65539 & 0xffff),
                        reversed = false,
                        msb      = false,
                        extended = false,
                        init     = 16w0x0000,
                        xor      = 16w0x0000) poly;
Hash<bit<16>>(HashAlgorithm_t.CUSTOM, poly) hash;
```

TOFINO

How has the CRC Custom been implemented in P4?

- **P4 language does not natively support the mod operation.**
- **CRC hard** … **nial mod.**
  - The T… **RC polynomials.**
  - MOD…




```
set_crc16_parameters
```

```
hash(
        nresult,
        HashAlgorithm.cr
        nbase,
        {ndata},ncount
);
```

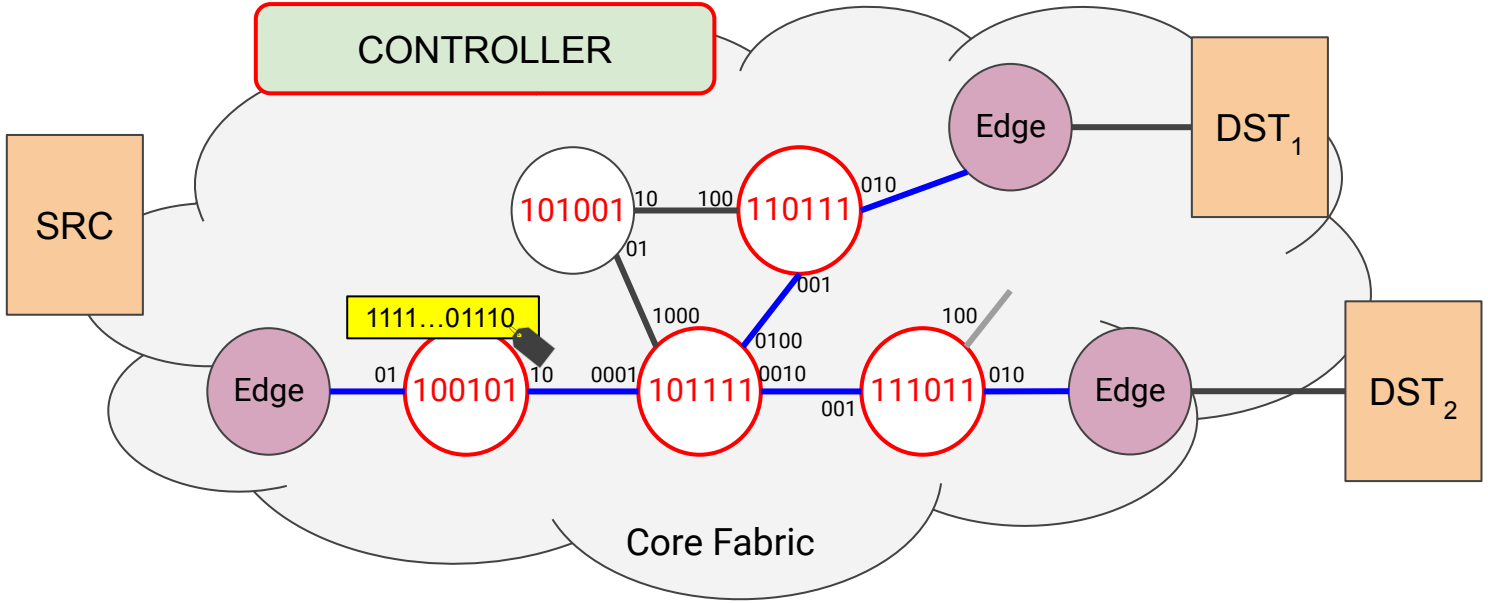
C interface

```
(65539 & 0xffff),
false,
false,
false,
16w0x0000,
16w0x0000) poly;
Hash<bit<16>>(HashAlgorithm_t.CUSTOM, poly) hash;
```

TOFINO

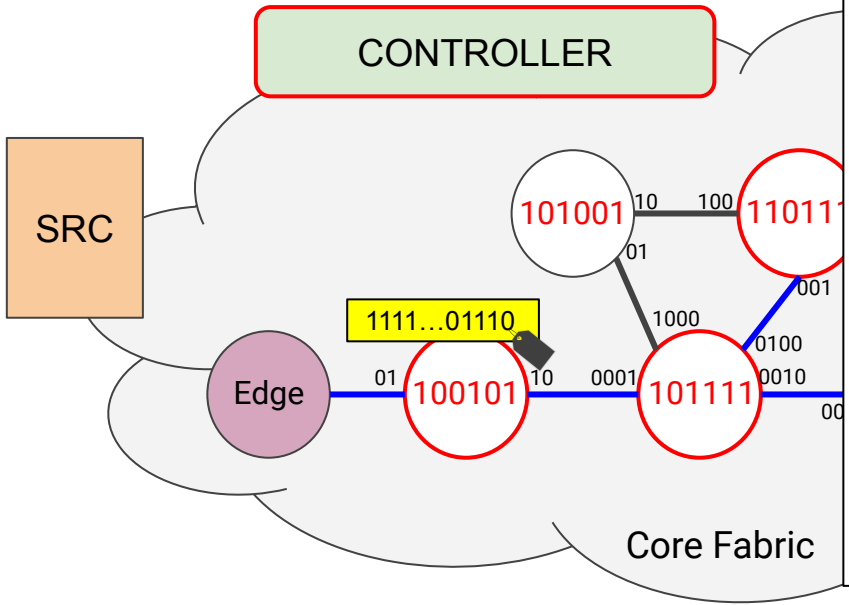
How does M-PolKA work?

- Forwarding using **mod** operation:
  - <111100001011101011 10>$_{100101}$ = 10 → steer to the port 2
- No *routeID* rewrite! No tables!



How does M-PolKA work?

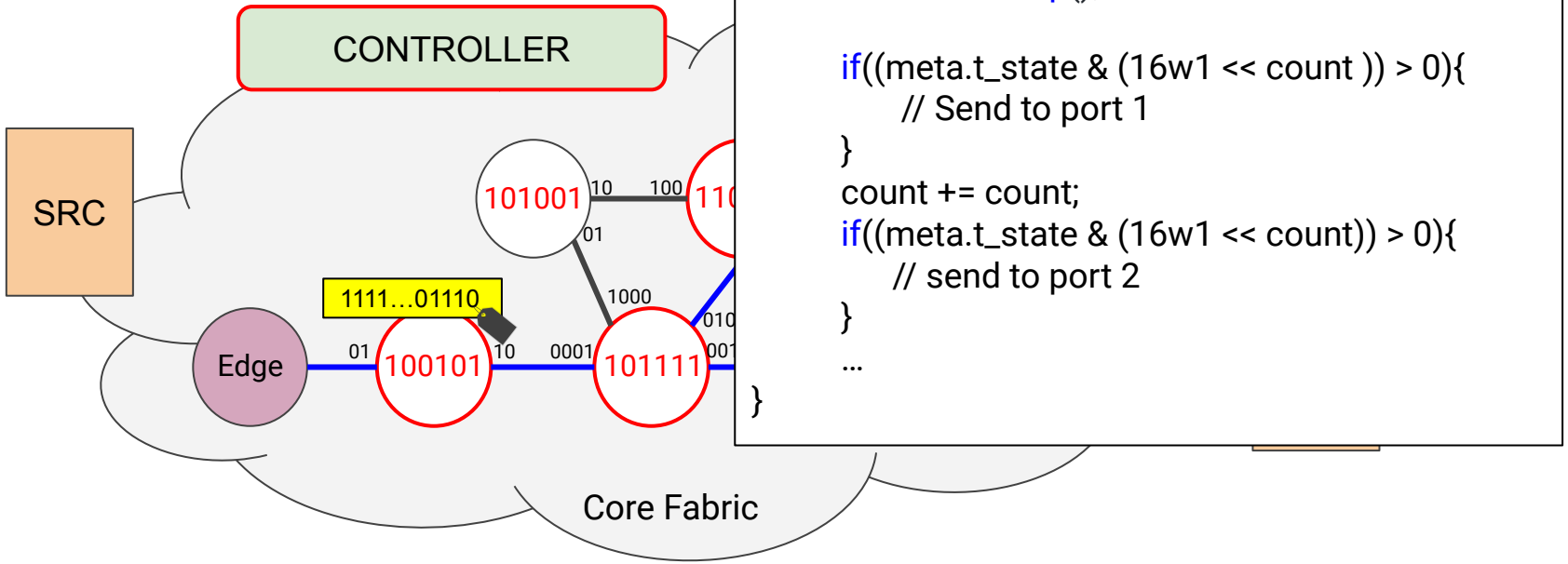
- Forwarding using **mod** operation:
  - <1111000010111010111 0>$_{100101}$ = 1
- No *routeID* rewrite! No tables!




```
action srcRoute_nhop() {
    bit<16> nbase = 0;
    bit<64> ncount = 4294967296*2;
    bit<16> nresult;

    bit<160>routeid = meta.routeId;

    bit<160> ndata = routeid >> 16;
    bit<16> dif = (bit<16>) (routeid ^ (ndata << 16));

    hash(
            nresult,
            HashAlgorithm.crc16_custom,
            nbase,
            {ndata},ncount
    );
    meta.t_state = nresult ^ dif;
}
```

How does M-PolKA work?

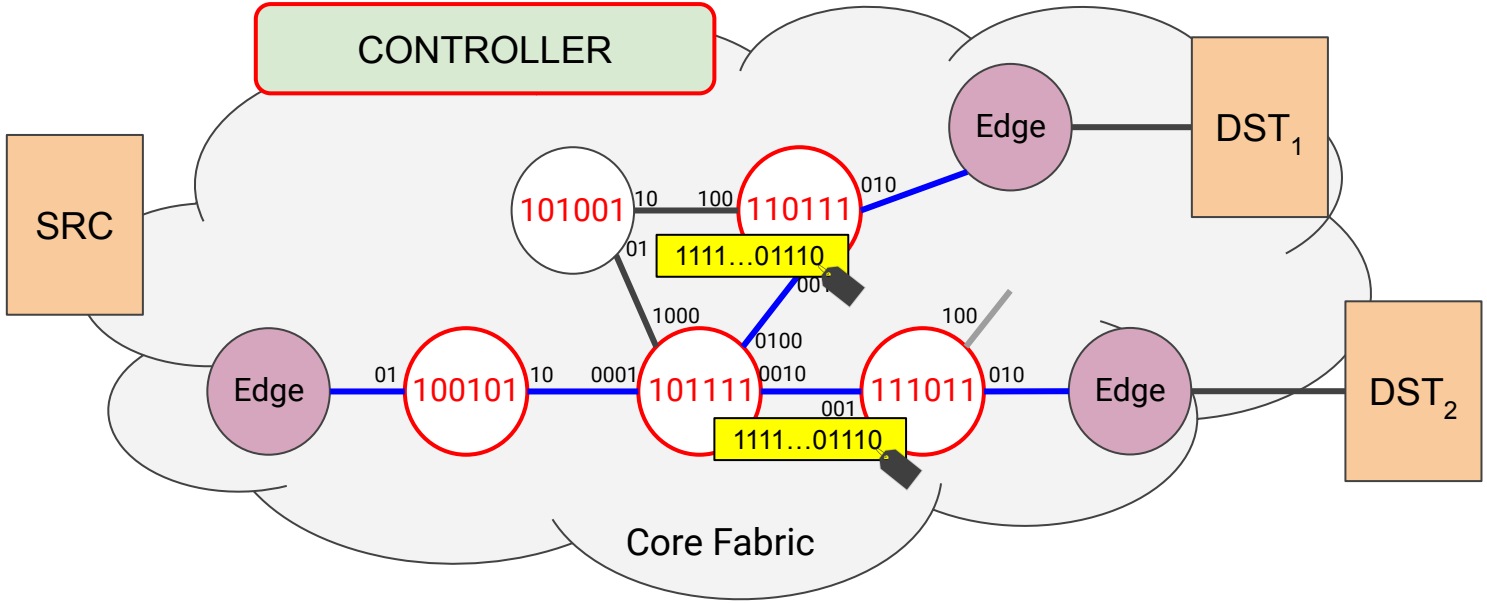
- Forwarding using **mod** operation:
  - <1111000010111010110>$_{100101}$
- No *routeID* rewrite! No tables!




```
apply {
        // Source-routing calculation
        srcRoute_nhop();

        if((meta.t_state & (16w1 << count )) > 0){
            // Send to port 1
        }
        count += count;
        if((meta.t_state & (16w1 << count)) > 0){
            // send to port 2
        }
        ...
}
```

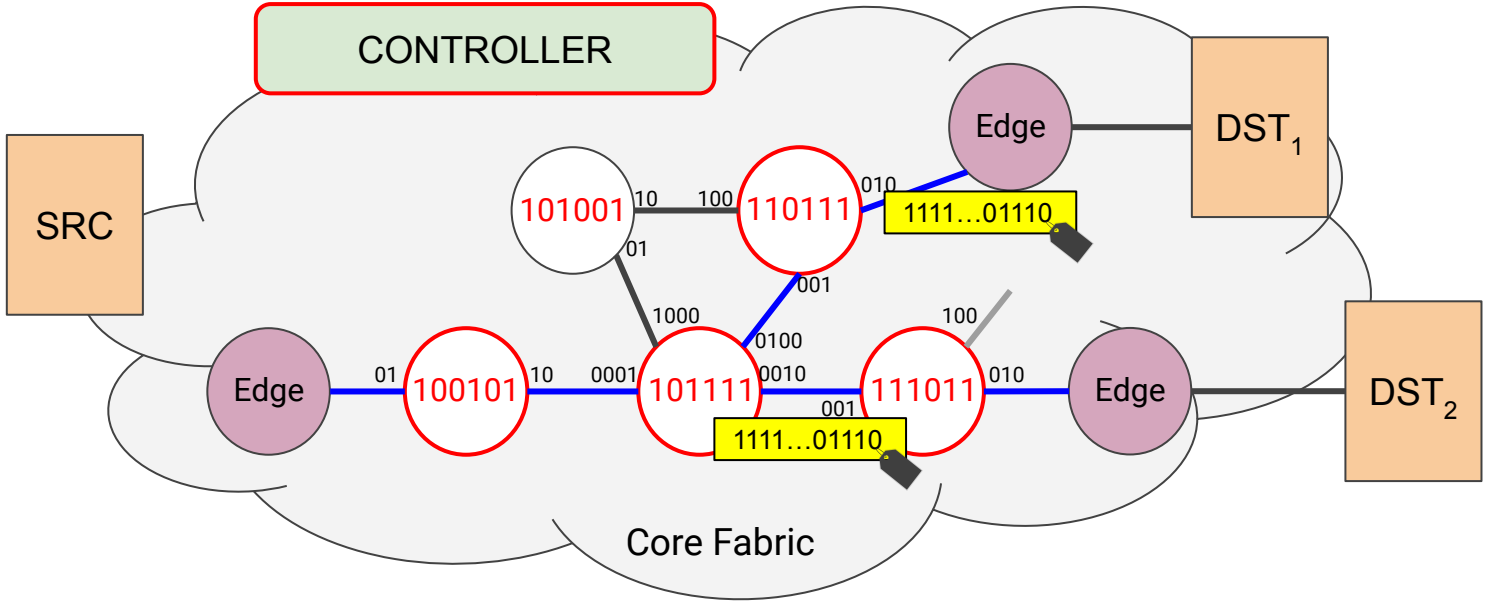
How does M-PolKA work?

- Forwarding using **mod** operation:
  - <1111000010111010111 0>$_{101111}$ = 110 → steer to the port 2 and 3
- No *routeID* rewrite! No tables!



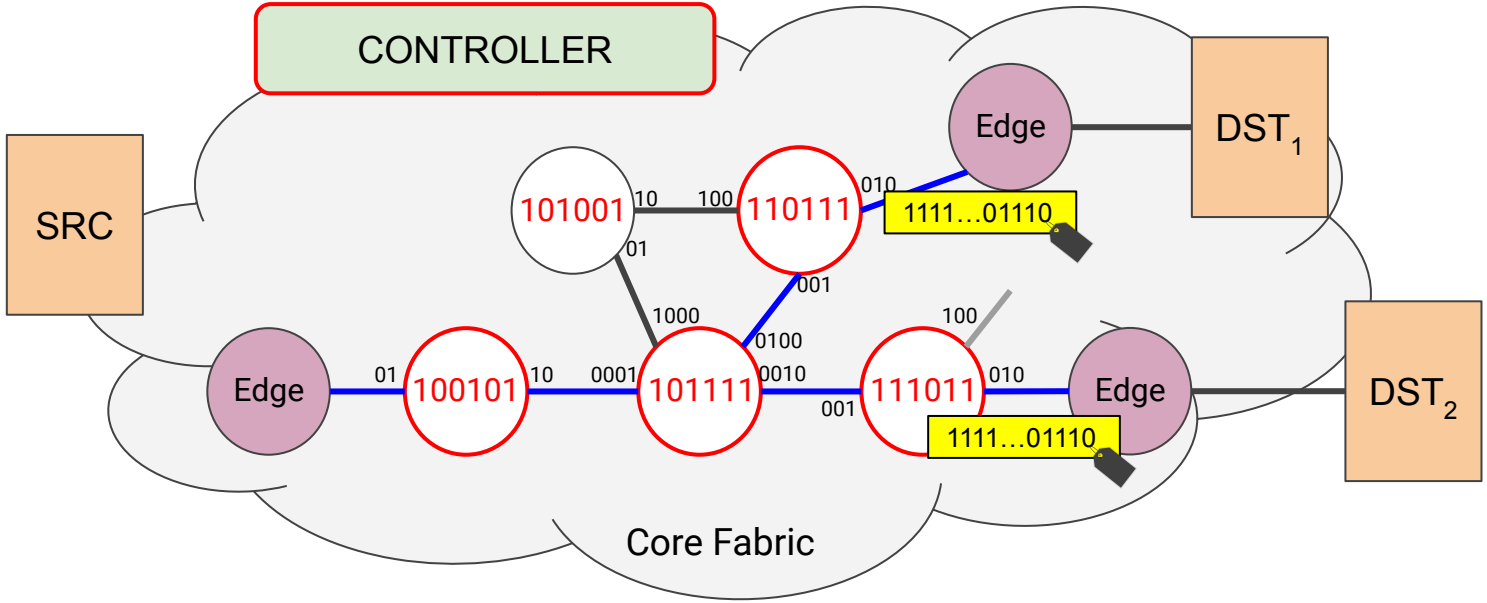
How does M-PolKA work?

- Forwarding using **mod** operation:
  - $<1111000010111010111 0>_{110111}$ = 10 → steer to the port 2
- No *routeID* rewrite! No tables!



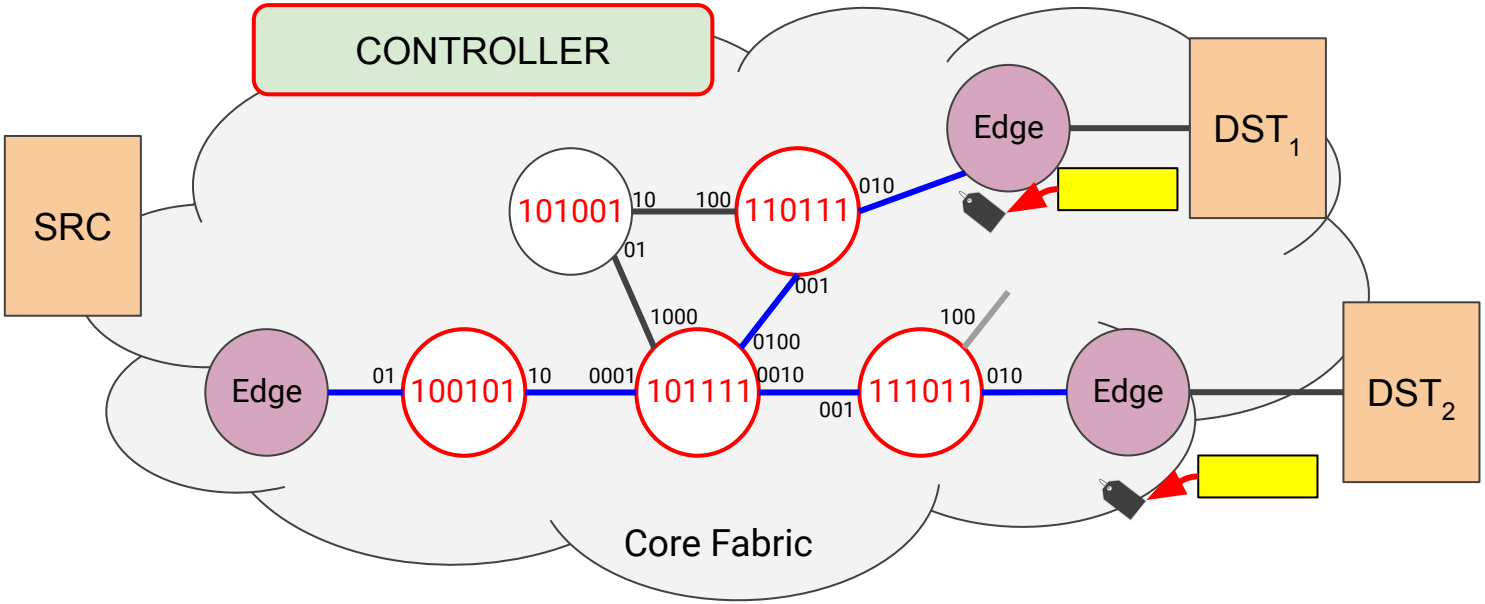
How does M-PolKA work?

- Forwarding using **mod** operation:
  - <111100001011101011 10>$_{111011}$ = 10 → steer to the port 2
- No *routeID* rewrite! No tables!



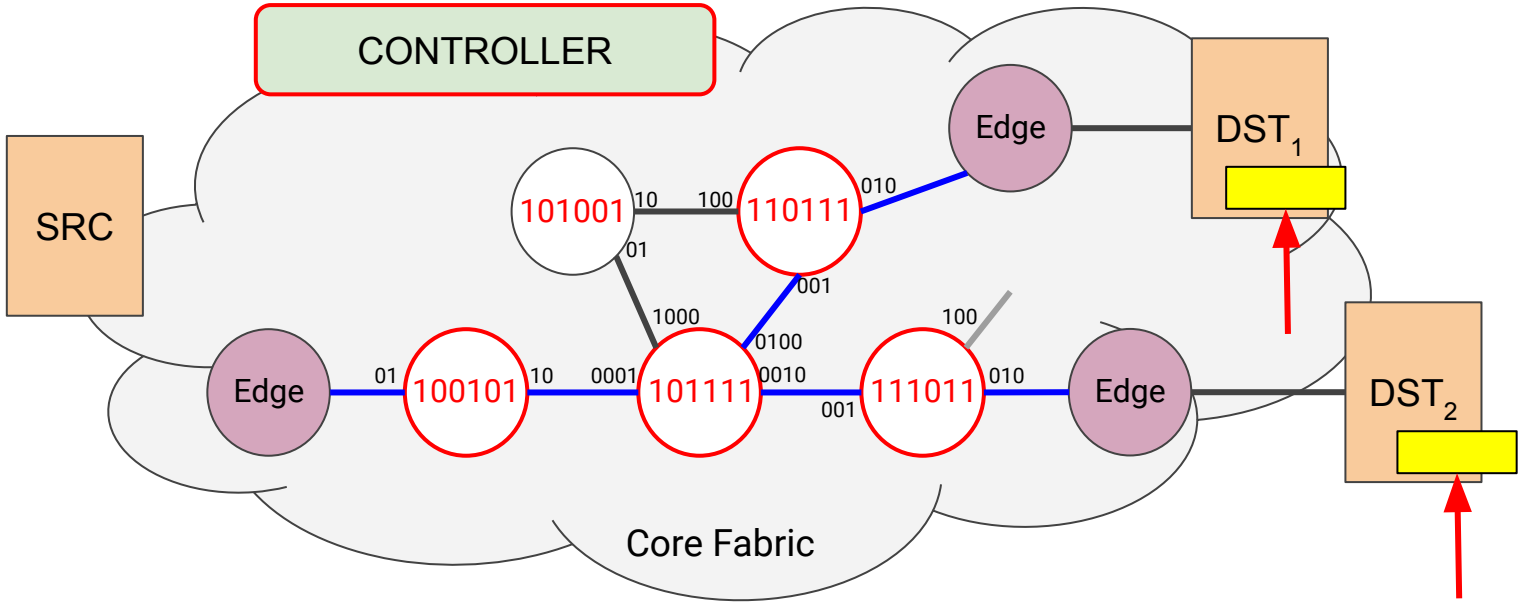
How does M-PolKA work?

- Finally, an action at edge egress node removes *routeID*.



How does M-PolKA work?

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

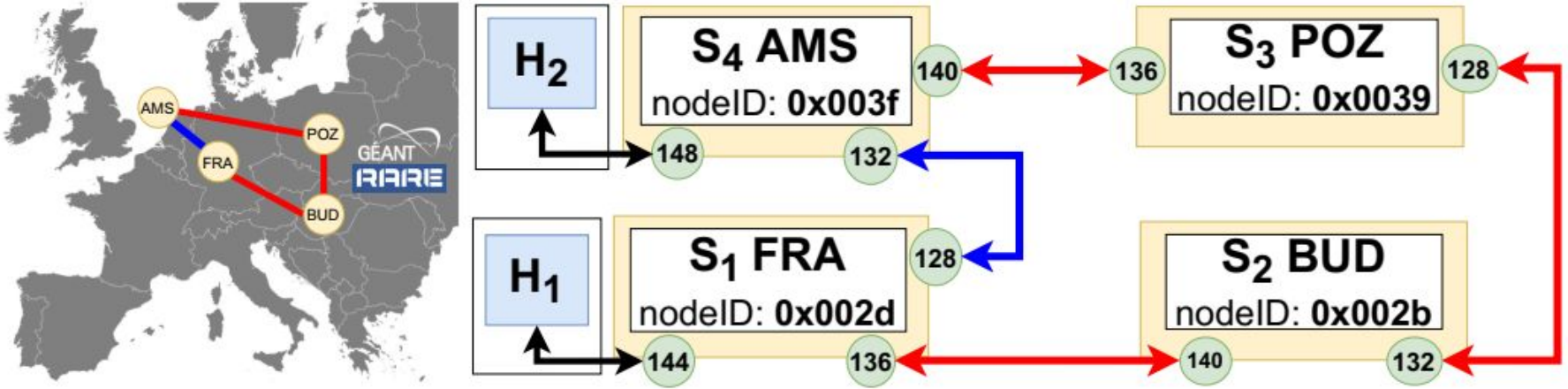


Agenda

- Motivation
- Proposal
- Design
- **Prototype**
- Conclusions
- Ongoing applications

M-PolKA: Data Plane Prototype

- M-PolKA is a generalization of our previous work, named PolKA.
- Our preliminary results were taken by using P4 in the bmv2 switch.
- However, we have PolKA's implementation for high-performance Tofino
  - Deployment: GEANT P4 Lab testbed
  - Preliminary results: ONDM 2021 conference paper



Integration of PolKA and M-PolKA into RARE project

- RARE: Open source full-featured router on open networking hardware for R&E
  - **data plane:** P4 bmv2 and DPDK (cpu-bound)
    - Tofino - only for PolKA
  - **control plane:** FreeRtr
    - Router OS process: it speaks various routing protocols, (re)encap packets, and exports forwarding tables to hardware switches.
    - It offers the first open implementation of Segment Routing.
  - Available a library to calculate the routeID written in Python
    - pip install polka-routing

Integration of PolKA and M-PolKA into RARE project

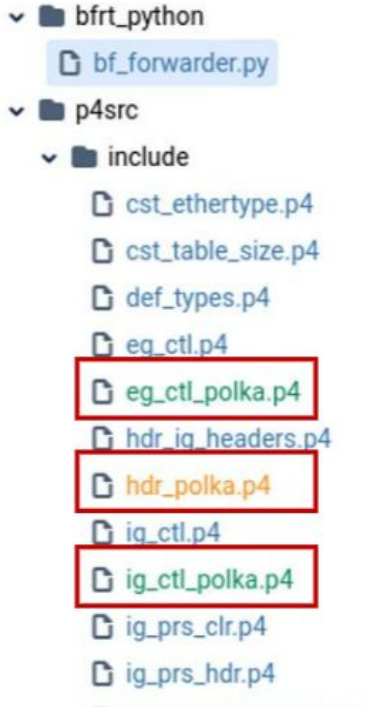
- **PolKA is the first non-standard protocol**
  - Thanks to Csaba Mate, and Frédéric Loui
    https://bitbucket.software.geant.org/projects/RARE/repos/rare/browse/p4src/include
    https://docs.freertr.net/guides/reference/

header

```
#ifndef _POLKA_P4_
#define _POLKA_P4_

#ifdef HAVE_POLKA
header polka_t {
    bit<8>          version;
    bit<8>          ttl;
    ethertype_t     proto;
    polka_route_t   routeid;
}
#endif

#endif // _POLKA_P4_
```

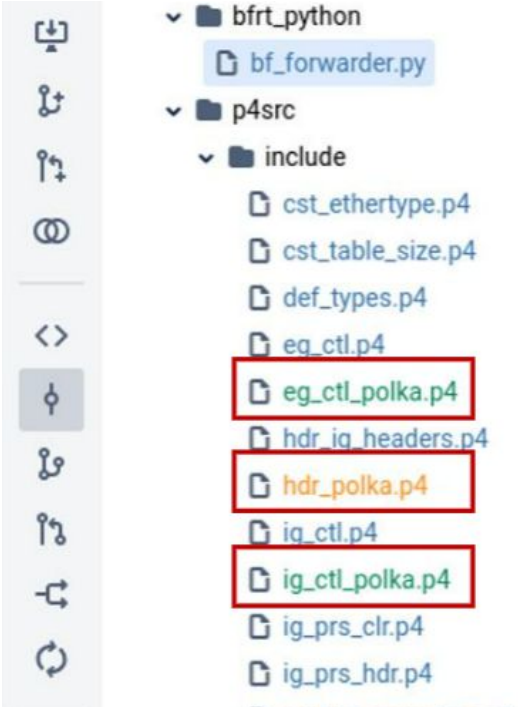
ingress control

```
CRCPolynomial<bit<16>>(
                        coeff    = (65539 & 0xffff),
                        reversed = false,
                        msb      = false,
                        extended = false,
                        init     = 16w0x0000,
                        xor      = 16w0x0000) poly;
Hash<bit<16>>(HashAlgorithm_t.CUSTOM, poly) hash;
```



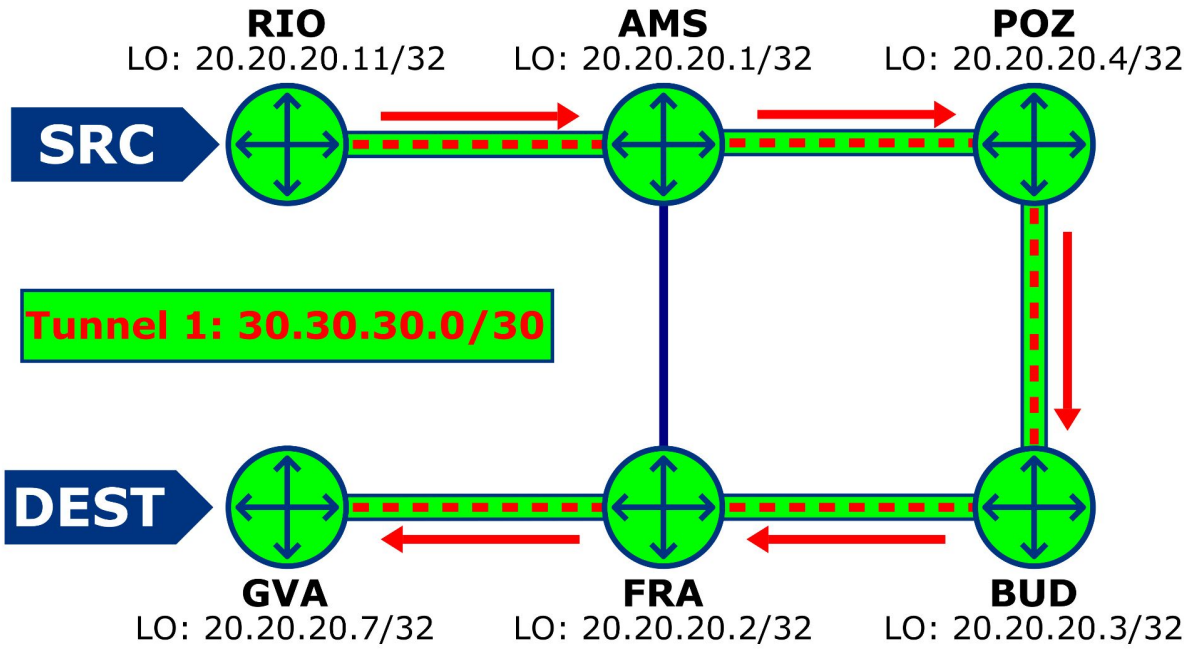
Integration of PolKA and M-PolKA into RARE project

- **PolKA is the first non-standard protocol**
  - Thanks to Csaba Mate, and Frédéric Loui
    https://bitbucket.software.geant.org/projects/RARE/repos/rare/browse/p4src/include
    https://docs.freertr.net/guides/reference/
- How to integrate **M-PolKA's control plane**?
  - Centralized Controller
  - **Reuse of standard distributed protocols**
    - Topology from link-state routing protocols
- **Fixed-length** M-PolKA header



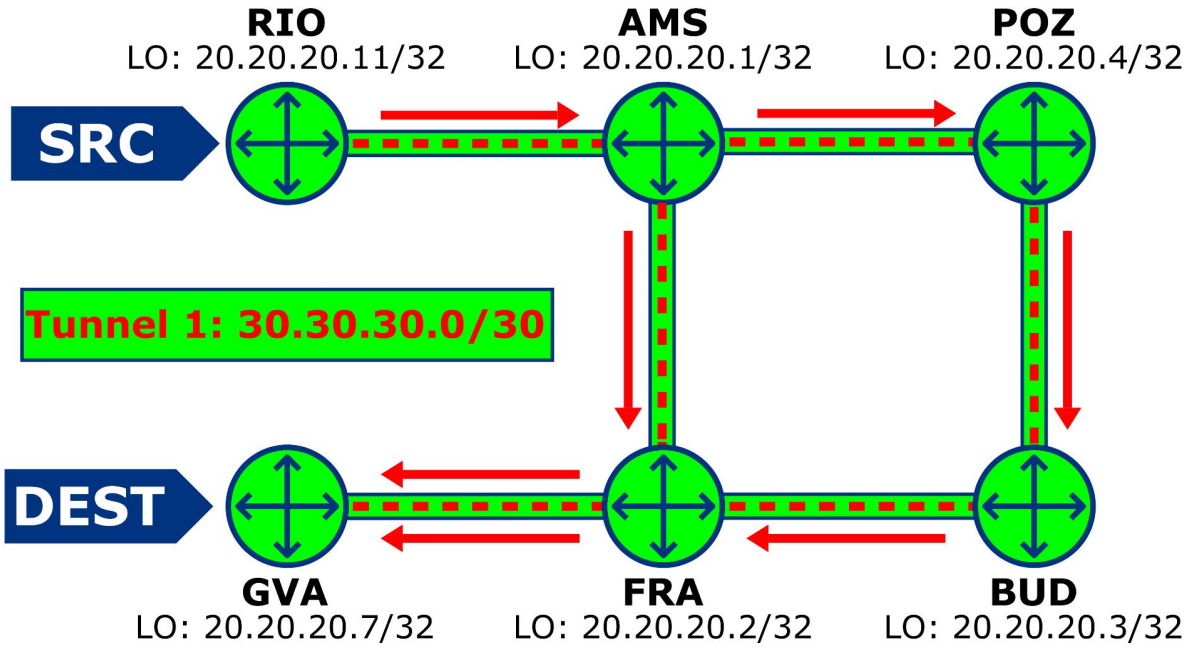
M-PolKA running into the FreeRtr

- Emulation in FreeRtr: Agile multipath reconfiguration in the RARE topology



M-PolKA running into the FreeRtr

- Emulation in FreeRtr: Agile multipath reconfiguration in the RARE topology



M-PolKA running into the FreeRtr

- M-PolKA configuration

```
RIO0001#show running-config interface ethernet1
interface ethernet1
 description RIO0001 --> AMS00001
 vrf forwarding v1
 ipv4 address 11.11.11.1 255.255.255.252
 template template1
 no shutdown
 exit
!
```

```
RIO0001#show running-config interface temp1
interface template1
 no description
 lldp enable
 vrf forwarding v1
 ipv4 address dynamic dynamic
 ipv6 address dynamic dynamic
 mpolka enable 11 65536 20
 mpls enable
 router lsrp4 1 enable
 router lsrp6 1 enable
 shutdown
 no log-link-change
 exit
!
```

nodeID

M-PolKA running into the FreeRtr

- M-PolKA configuration

```
RIO0001#show running-config interface temp1
interface template1
 no description
 lldp enable
 vrf forwarding v1
 ipv4 address dynamic dynamic
 ipv6 address dynamic dynamic
 mpolka enable 11 65536 20
 mpls enable
 router lsrp4 1 enable
 router lsrp6 1 enable
 shutdown
 no log-link-change
 exit
!
```

node index maps to nodeID polynomial

```
RIO0001#show mpolka routeid tunnel1
iface      hop          routeid
ethernet1  11.11.11.2   00 00 00 00 00 00 08 4e b4 2e eb 6f 98 63

index  coeff     poly   crc    equal
0      00010000  53098  53098  true
1      00010001  6      6      true
2      00010003  4      4      true
3      00010005  2      2      true
4      00010009  4      4      true
5      0001000f  28491  28491  true
6      00010011  21399  21399  true
7      0001001b  1      1      true
8      0001001d  23146  23146  true
9      0001002b  50980  50980  true
10     0001002d  42850  42850  true
11     00010039  51279  51279  true
12     0001003f  40235  40235  true
13     00010047  28851  28851  true
14     0001004b  53417  53417  true
15     00010053  14915  14915  true
16     00010059  22787  22787  true
17     00010063  48965  48965  true
18     00010065  8332   8332   true
19     0001006f  6301   6301   true
```

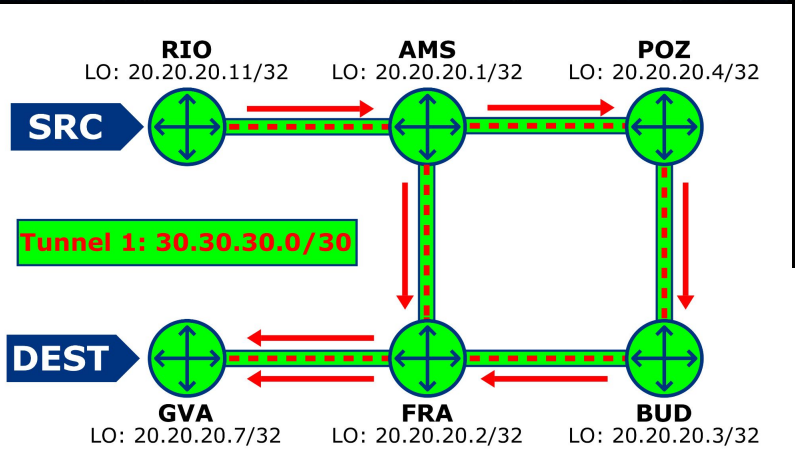
routeID of tunnel1

M-PolKA running into the FreeRtr

- M-PolKA tunnel creation

multiple paths

```
RIO0001#show running-config interface tunnel1
interface tunnel1
 description MPOLKA tunnel from RIO0001 -> GVA0001
 tunnel vrf v1
 tunnel source loopback0
 tunnel destination 20.20.20.7
 tunnel domain-name 20.20.20.1 20.20.20.2 20.20.20.4 , 20.20.20.2 20.20.20.7 , 20.20.20.3 20.20.20.2 , 20.20.20.4 20.20
.20.3 , 20.20.20.7 20.20.20.7 ,
 tunnel mode mpolka
 vrf forwarding v1
 ipv4 address 30.30.30.1 255.255.255.252
 no shutdown
 no log-link-change
 exit
!
```

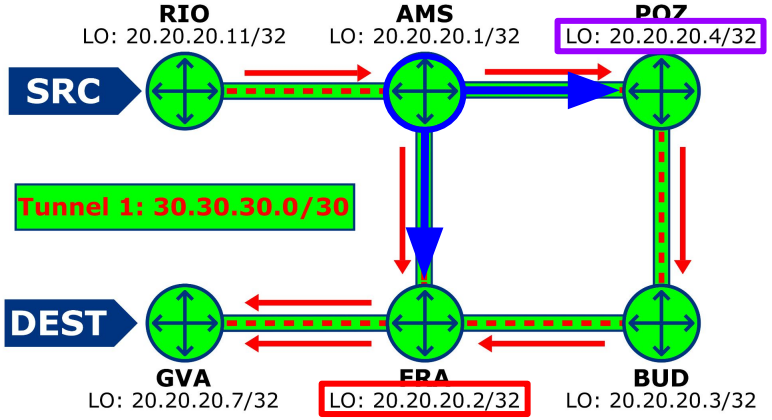


M-PolKA running into the FreeRtr

- M-PolKA tunnel creation

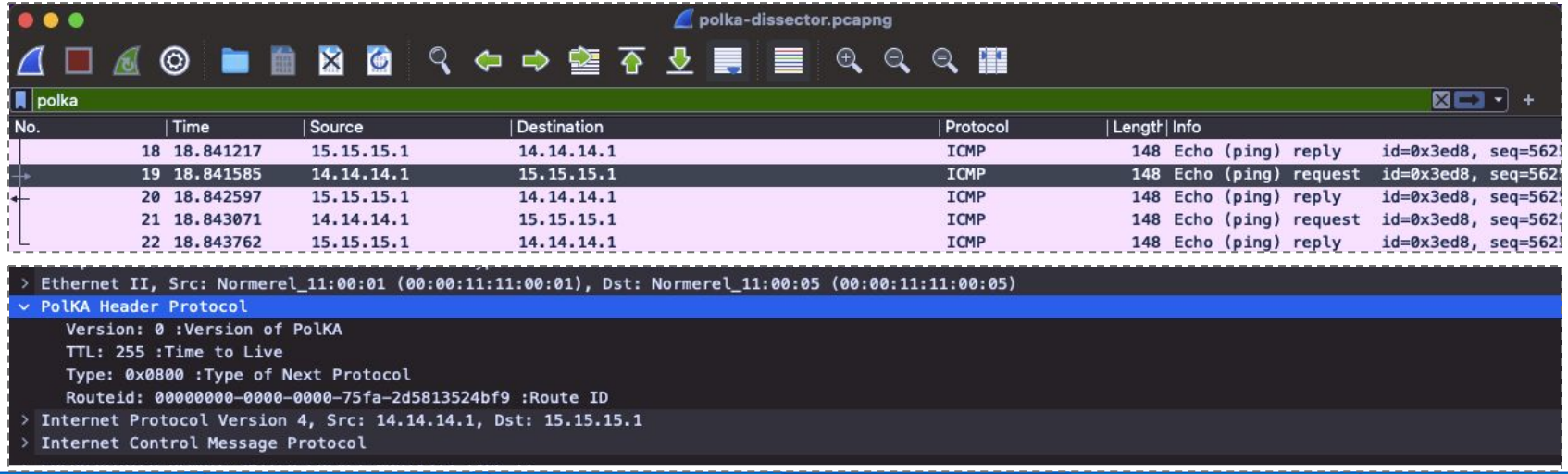
multiple paths

```
RIO0001#show running-config interface tunnel1
interface tunnel1
 description MPOLKA tunnel from RIO0001 -> GVA0001
 tunnel vrf v1
 tunnel source loopback0
 tunnel destination 20.20.20.7
 tunnel domain-name 20.20.20.1 20.20.20.2 20.20.20.4 , 20.20.20.2 20.20.20.7 , 20.20.20.3 20.20.20.2 , 20.20.20.4 20.20
.20.3 , 20.20.20.7 20.20.20.7 ,
 tunnel mode mpolka
 vrf forwarding v1
 ipv4 address 30.30.30.1 255.255.255.252
 no shutdown
 no log-link-change
 exit
!
```



PolKA and M-PolKA: Github

- **https://github.com/nerds-ufes/polka/**
  - References
  - Tutorials (Mininet and FreeRouter)
  - Wireshark dissector
  - More to come…



Agenda

- Motivation
- Proposal
- Design
- Prototype
- **Conclusions**
- Ongoing applications

Conclusions

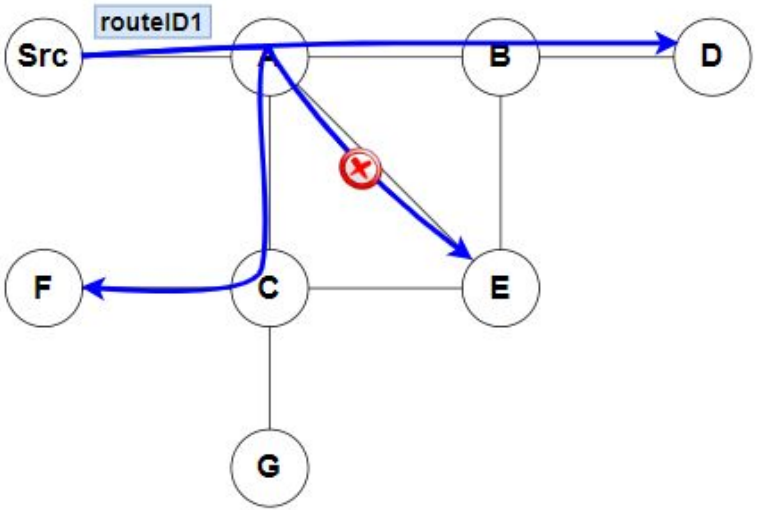
- It is **feasible to deploy M-PolKA in high-performance programmable network devices** by reusing CRC hardware.
  - Preliminary results: By using CRC hardware we are able to match the performance of traditional solutions
- Potential to enable a new range of complex network applications.
  - Current works:
    - a full P4 Tofino implementation
    - control plane dependability and applications

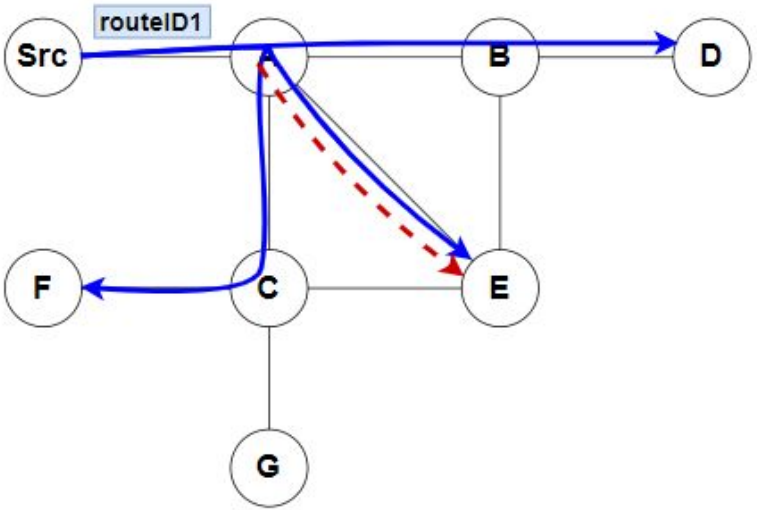
Agenda

- Motivation
- Proposal
- Design
- Prototype
- Conclusions
- **Ongoing applications**

Ongoing applications: Agile Multicast Reconfiguration

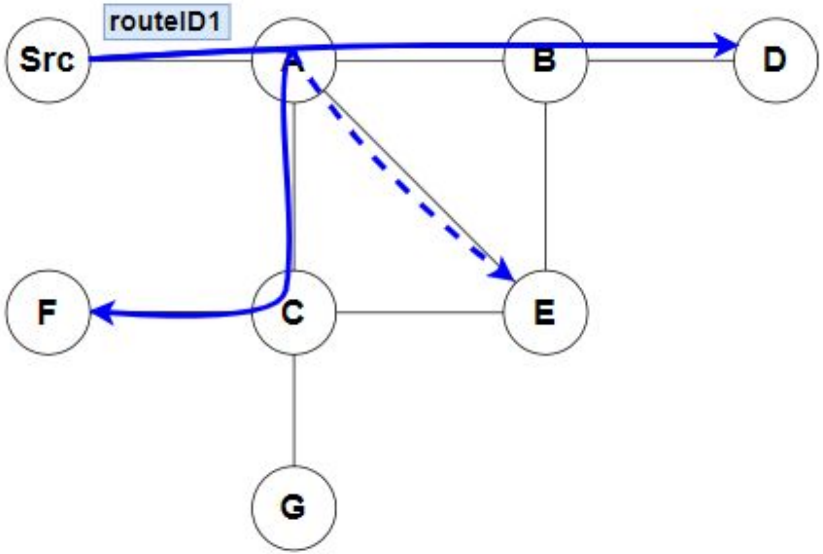
- Example: Reconfigure branch in Multicast tree
  - Failure reaction
  - QoS
  - New client to deliver content

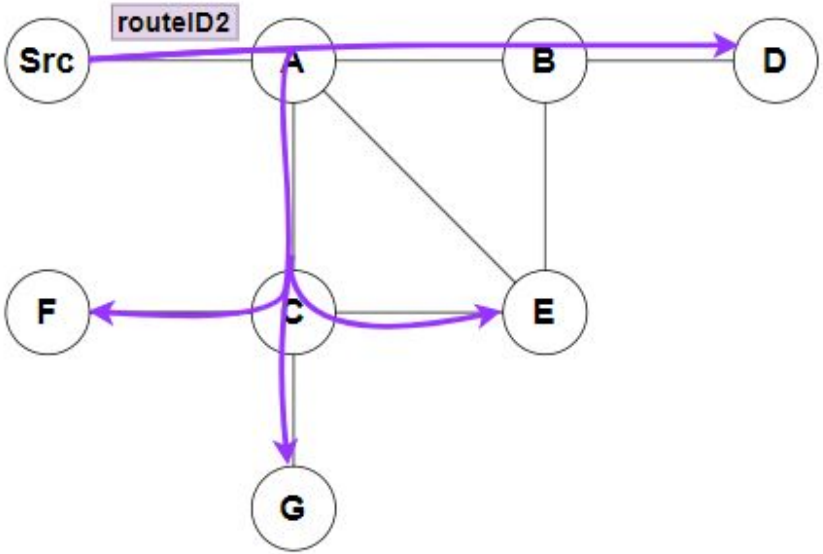




Ongoing applications: Agile Multicast Reconfiguration

- Agile modification of branches:
  - Add G + Change path to E (via C)
- A single entry at the edge:
  - Packets are tagged with the new *routeID*.





Thank you!

Rafael Silva Guimarães

rafaelg@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.*