Source Routing on the Edge

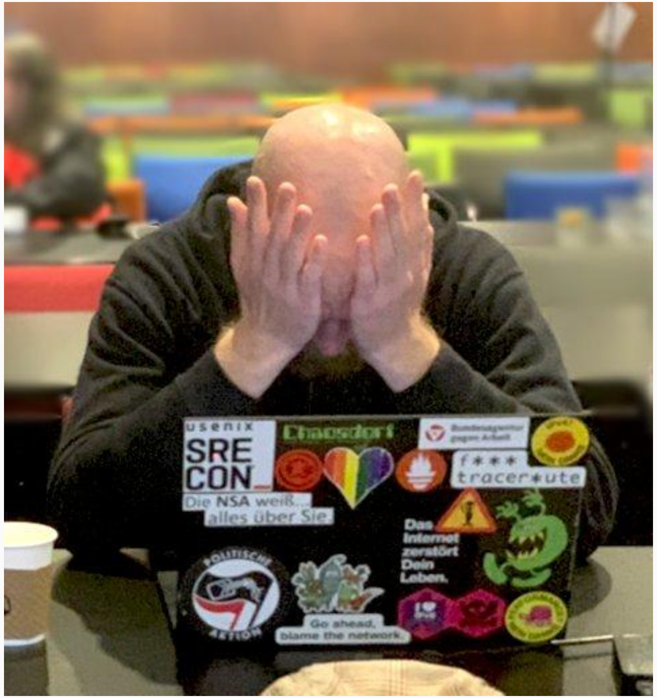
Scale, Reliability and Programmability for EXARINGs Internet Peering

Agenda

1. Who am I
2. State of Packet Forwarding
3. Requirements of modern Packet Forwarding
4. Solution
    a. Data Plane
    b. Control Plane
    c. Issues
5. Questions

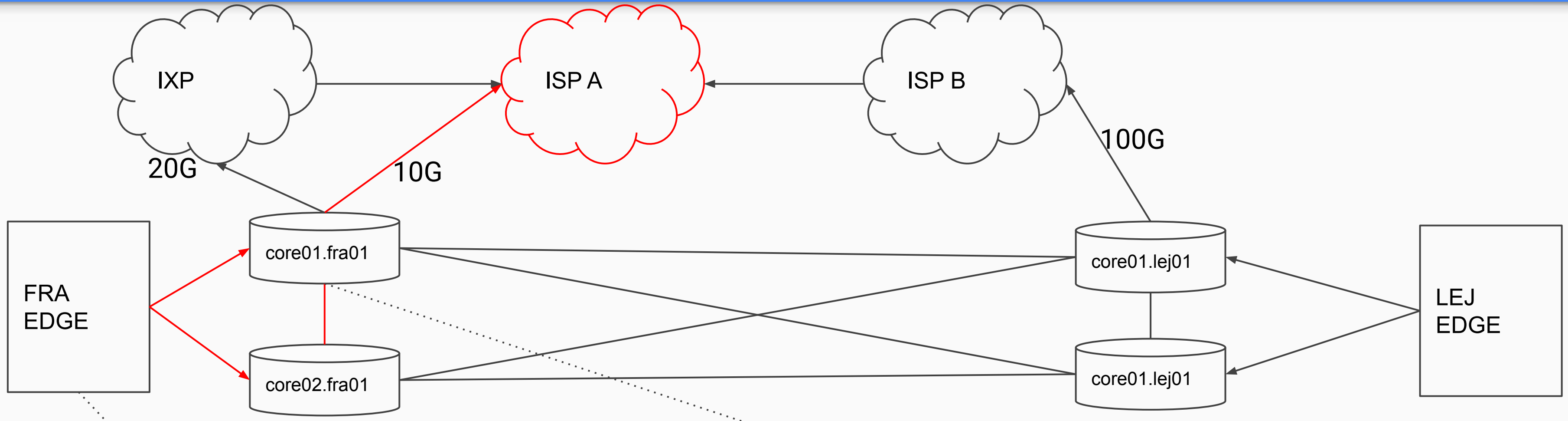
Who am I?

- ➔ Oliver Herms aka takt
- ➔ Senior Network Engineer @ EXARING AG
- ➔ Friend of robustness, reliability, velocity
- ➔ Network Automation Enthusiast
- ➔ Golang and gRPC fanboy



State of Packet Forwarding

State of Packet Forwarding



Dest.	Next-Hop	Weight
Any	core01.fra01	-
Any	core02.fra01	-

Dest.	Next-Hop	Priority	Weight
ISP A	ISP A	1000	100%
ISP A	IXP	500	0%
ISP A	core01.lej01	100	0%

Limitations of current state (1)

- ➔ Packets to an ISP follow a single shortest path or a number of equal paths
  - All active links get the same amount of traffic
  - 10G + 100G = 20G usable capacity
  - What is equal can be tuned administratively

Limitations of current state (2)

Traffic Engineering can make use of non-shortest paths

- ➔ Manual tweaking of Route attributes
  - ◆ Dangerous: Mistakes can cause outages
- ➔ Only on a per Prefix basis (IP Ranges, 256-2M addresses)
- ➔ Requires changes in Router configs
  - ◆ We fully generate them. But we review them manually.

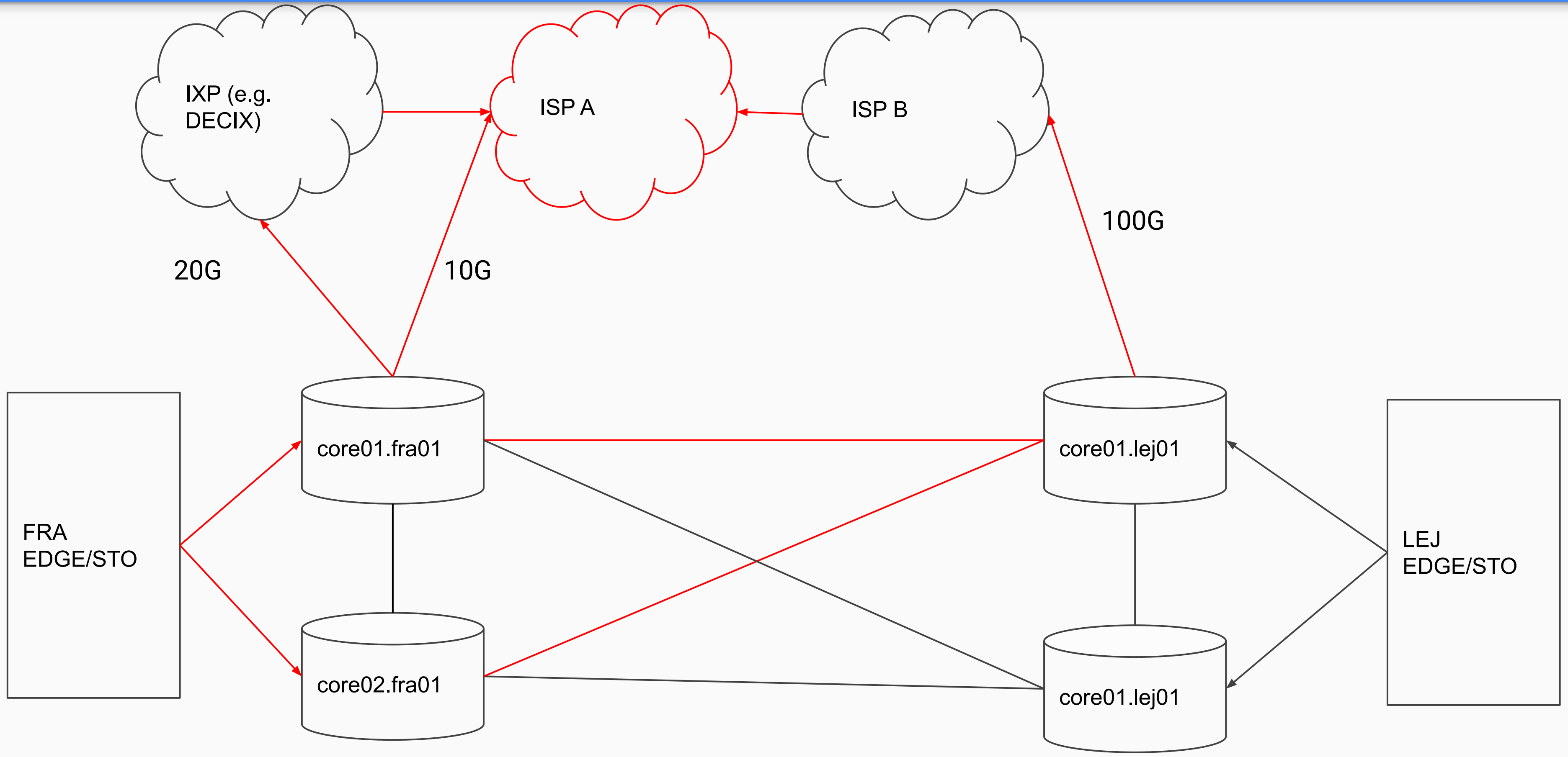
Limitations of current state (3)

All IP Routes must be installed into Routers

- ➔ Memory is limited
- ➔ Expensive licenses required for 100k+ Routes
- ➔ Limits future growth with current platform
- ➔ Stops us from using even cheaper Routers

Requirements

Requirements



Requirements

- ➔ Make non-equal speed links usable
- ➔ Make non-equal cost links usable
- ➔ Automatically maximize utilization of cheapest links
- ➔ Automatically move excess traffic to next cheapest link
- ➔ Allow to take link quality into account in routing decision
- ➔ React to changes quickly and repair any situation automatically, if possible

Nice to haves

- ➔ Do not change Router configs
- ➔ Support arbitrary amount of Routes
- ➔ Allow per IP traffic engineering

Solution

Solution (1)

- ➔ Let Vendor Routers forward traffic but not route it
    - ◆ Too inflexible to meet our needs
- ➔ Source Routing: Let the source of traffic decide which path a packet takes
- ➔ Servers send labeled packets
- ➔ Packets get encapsulated into tunnels to Egress Routers

Solution (2)

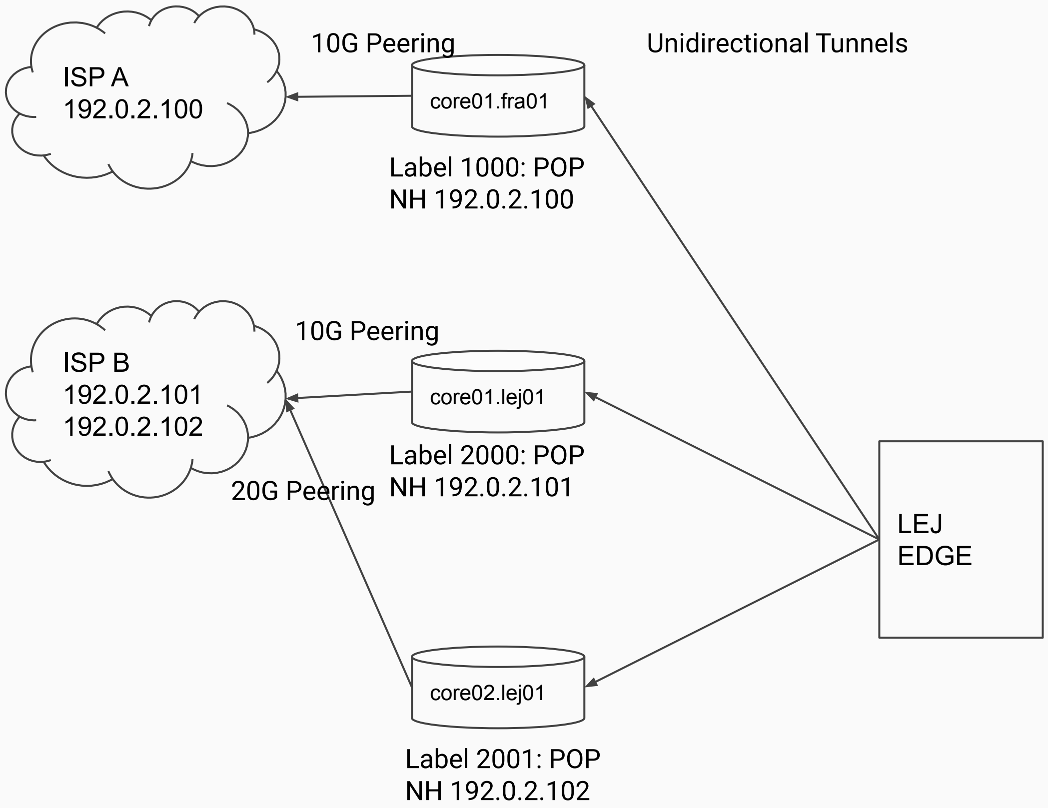
- Labeled packet arrives at Router
  - Static forwarding
  - Label indicates next-hop
  - Ignoring IP Routing Table

Advantages

- ➔ Allows fine granular control of link utilization
  - will save € in OPEX
- ➔ No need for IP Routing on Routers anymore
  - will save € in CAPEX)

Data Plane

Architecture Overview (Data Plane)



Dest.	Label	Tunnel	Weight
ISP A	1000	core01.fra01	10G
ISP B	2000	core01.lej01	10G
ISP B	2001	core02.lej01	20G

MPLS Label Switching Paths

- Multiprotocol Label Switching (MPLS)
- Label Switching Path (LSP) allows choosing Next-Hops per Label

```
oherms@core02.fra01> ...nces CLOSEDNET protocols mpls static-label-switched-path coffee_62_69_146_95
transit 1001899 {
    description rdev=AS201701,rif=ECIX-FRA,ndev=ECIX-FRA,nif=ECIX-FRA-001,nrole=IXP;
    next-hop 62.69.146.95;
    pop;
}
```

Getting to the Peering Router (PR)

- Full MPLS deployment on internal network
  - IS-IS SR (Segment Routing)
  - LDP (Label Distribution Protocol)
  - RSVP (Resource Reservation Protocol)
- MPLS in a Tunnel
  - MPLS over GRE/IP
  - MPLS over UDP/IP

Packet Stack leaving Machines

Tunnel IP Header	UDP Header Port 6635	MPLS Label (Next Hop)	IP Header of Payload	TCP/UDP Header	Data...

Machines (Linux)

1. Create Foo Over UDP (FOU) encapsulated SIT tunnel per Router

```
# modprobe fou
# ip fou add port 6635 ipproto 4
# ip link add name cn-cr01fra01-0 type sit remote 192.168.1.1 local
192.168.1.2 ttl 64 encap fou encap-sport 6635 encap-dport 6635
```

2. Add MPLS encapsulated tunnel interface routes

```
# modprobe mpls_iptunnel
# modprobe mpls_gso
# ip route add 192.0.2.0/24 encap mpls 123 dev cn-cr01fra01-0
```

Router Tunnel Endpoints

Decap MPLS-in-UDP Firewall Filter

```
oherms@core02.fra01> show configuration firewall family inet filter
CN_MATROSCHKA
term MPLS-IN-UDP {
    from {
        destination-prefix-list {
            CN_MATROSCHKA_CORE02_FRA01_v4;
        }
        protocol udp;
        destination-port 6635;
    }
    then {
        decapsulate mpls-in-udp;
    }
}
...
```

Control Plane

Requirements (1)

- Calculate routing view per Region
    - All machines in a region should have identical routing tables

Requirements (2)

- ➔ Reliable
  - Must survive machine failure
  - Must support In Service Software Update (ISSU, no it’s not a trap)

Requirements (3)

- Scalable
  - Must support 100+ clients per Region
  - Growing Internet Routing Tables
  - Growing number of Peerings

Requirements (4)

- Programmable
  - Allow administrative changes to default routing decisions

Getting Routes from Routers

Make BMP Data usable

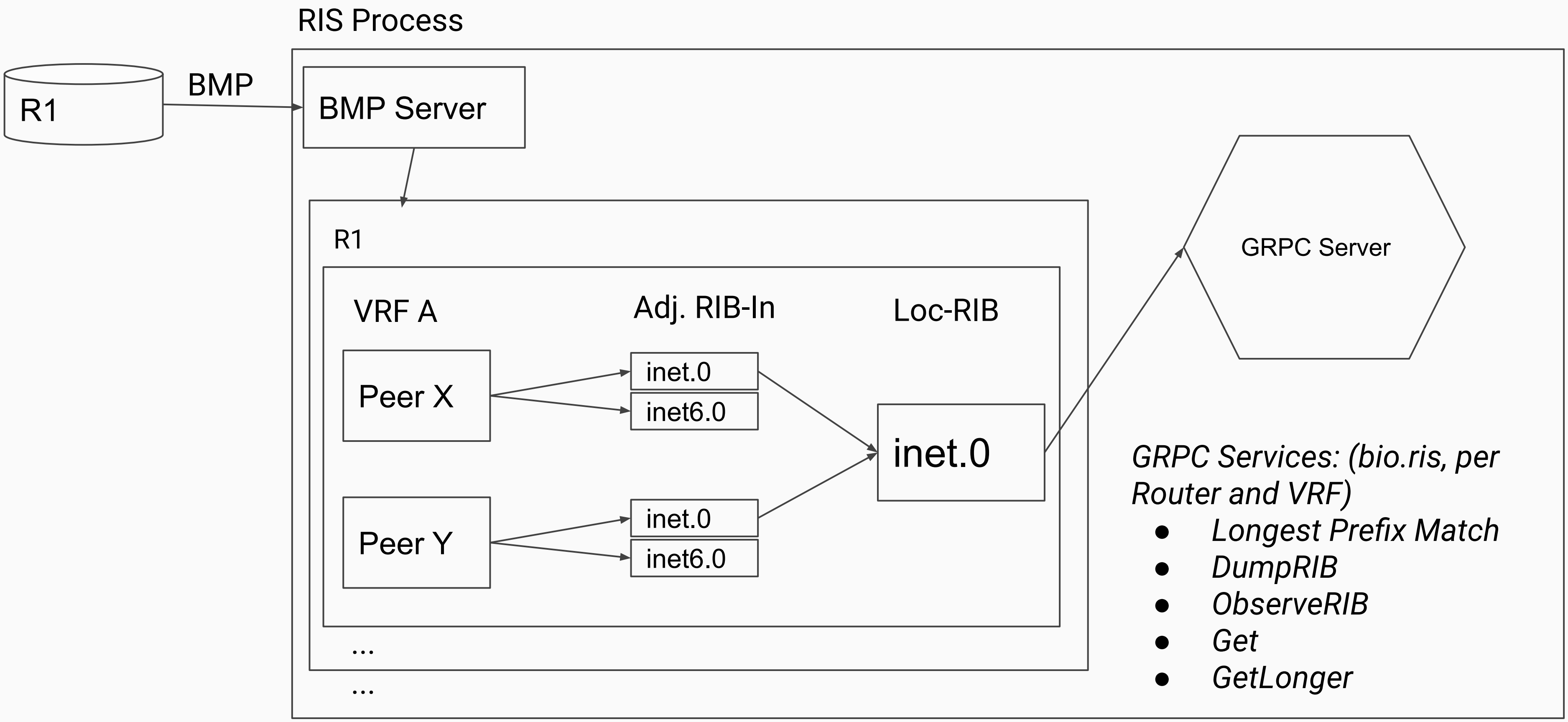
Getting Routes from Routers (1)

- BGP Monitoring Protocol (BMP, RFC 7854)
  - Sends all received routes to a monitoring station
  - Notifies monitoring station about peer up/down events
  - Either pre-policy or post-policy
    - We use post-policy

Getting Routes from Routers (2)

- BIO-Routing Route Information Service (RIS)
  - github.com/bio-routing/bio-rd/cmd/ris
  - Receives BMP messages
  - Tracks per Router/VRF/Peer Adj-RIB-In State
  - Exposes state via gRPC

Getting Routes from Routers (3)



Getting Routes from RIS into SDN Controller

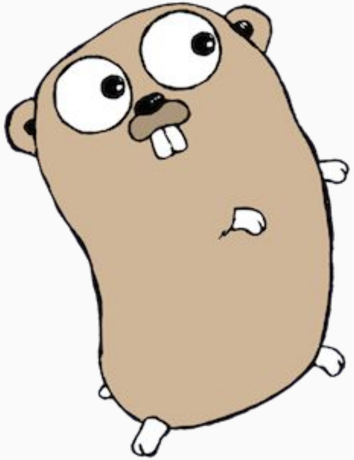
- Route Information Service (RIS) allows streaming routing information per Router/VRF
- Uses gRPC Streaming RPC
  - Call ObserveRIB()
  - Reads an (endless) stream of updates
  - RIS sends a state dump initially + updates as they come in via BMP

SDN Controller

Decision Making

Route Controller / SDN Controller (1)

- Written in Go
- Discovers MPLS Label to Next Hop mapping from IPAM
- Calculates shortest paths based on BGP data
  - Per Region
  - Per Prefix
  - BGP Attributes:
    - Local Pref
    - Autonomous System Path
    - MED
    - Origin
    - Internal cost to Next-Hop



Route Controller / SDN Controller (2)

- ➔ Takes Traffic Engineering Input
  - Allows overriding BGP path information
  - To be done automatically
  - Manual action for now

Route Controller / SDN Controller (3)

- Traffic Engineering Controller is under development
  - Multi-Instance
  - Single leader
  - Takes input from
    - OpenConfig Streaming Telemetry
    - Netflow Collector (tflow2)
    - RIS

Route Controller / SDN Controller (4)

- ➔ Streams Routing Tables to Machines
- ➔ gRPC Streaming RPC
- ➔ New clients receive a full dump
- ➔ Incremental updates sent as route decisions change

Route Attributes:

- Prefix
- Exit Routers Tunnel IP-Address
- MPLS Label
- Weight

Route Agent

Getting Routes into Machines

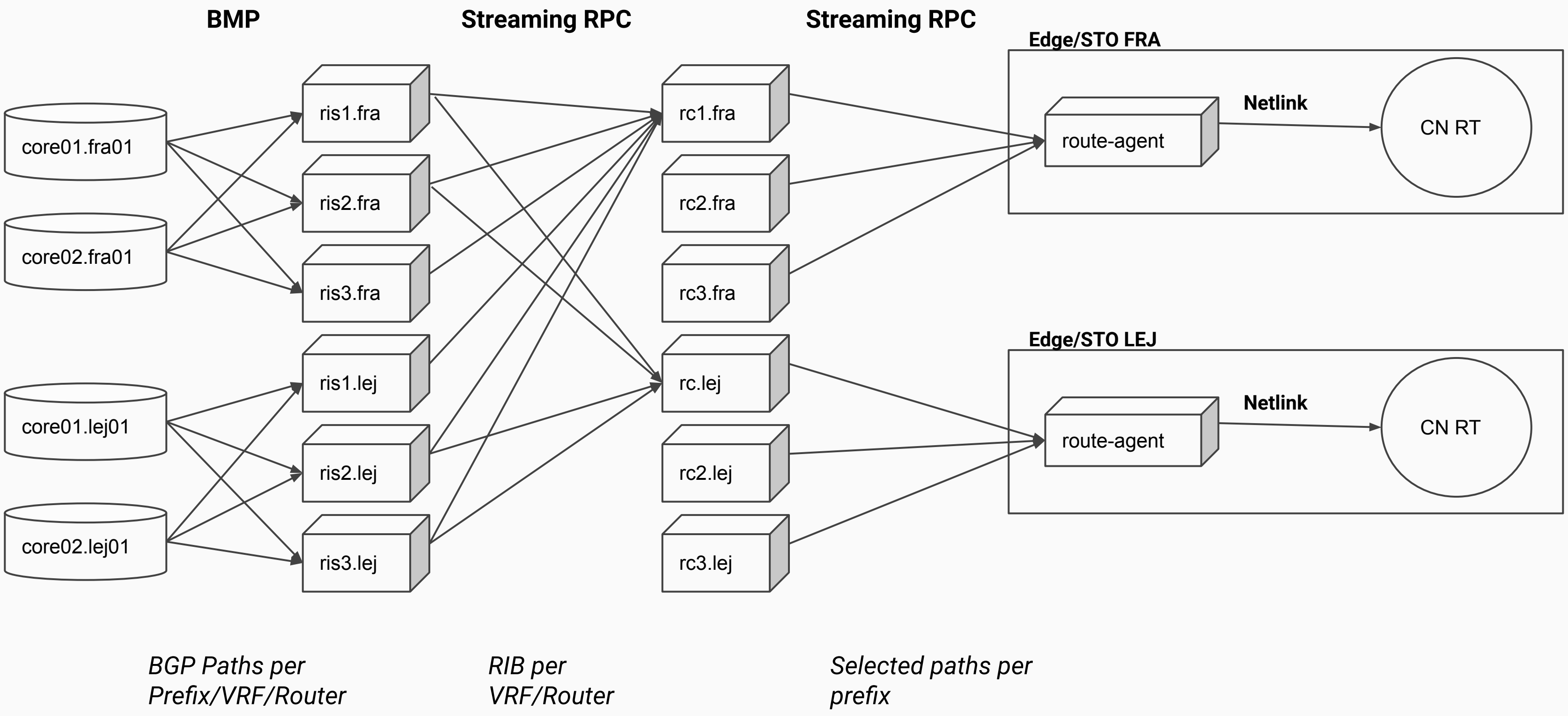
Route Agent (1)

- Written in Go
- Makes sure necessary Kernel Modules are loaded
  - fou
  - mpls_iptunnel
  - mpls_gso

Route Agent (2)

- ➔ Configures Tunnels to Routers
  - ◆ Routers are being discovered from Datacenter Inventory Service
- ➔ Maintains a Machines Routing Table
  - ◆ Receives Updates from Route Controller
  - ◆ Uses Netlink to Replace/Delete Routes in the Linux Kernel

Architecture Overview (Control Plane)



Issues encountered

Go/Netlink issue

- ➔ github.com/vishvananda/netlink
- ➔ Unable to write Multipath Routes with MPLS Encap into the Kernel
- ➔ Encap attribute attached to the wrong object
- ➔ Pull Request waiting for merge

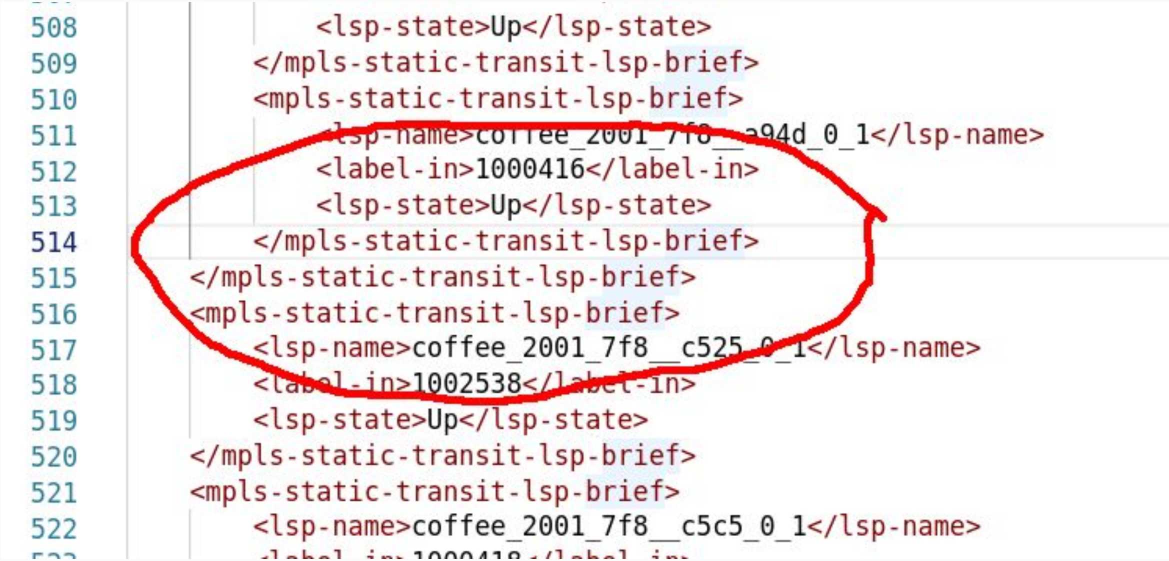
Vendor BMP Issue

- ➔ Router sends incomplete BGP OPEN messages in BMP Peer Up Notifications
- ➔ Only when the peer Router sends exactly 4 Byte-ASN and AddPath capabilities
- ➔ Only when using “allow-from” instead of “neighbor” statement
- ➔ BGP OPEN optional parameters missing

Vendor CLI Output Issues

Showing static LSPs briefly as XML output results in invalid XML

- ➔ Only with 100+ LSPs configured
- ➔ JSON output causes segfault




```
            <lsp-state>Up</lsp-state>
        </mpls-static-transit-lsp-brief>
        <mpls-static-transit-lsp-brief>
            <lsp-name>coffee_2001_7f8__a94d_0_1</lsp-name>
            <label-in>1000416</label-in>
            <lsp-state>Up</lsp-state>
        </mpls-static-transit-lsp-brief>
    </mpls-static-transit-lsp-brief>
    <mpls-static-transit-lsp-brief>
        <lsp-name>coffee_2001_7f8__c525_0_1</lsp-name>
        <label-in>1002538</label-in>
        <lsp-state>Up</lsp-state>
    </mpls-static-transit-lsp-brief>
    <mpls-static-transit-lsp-brief>
        <lsp-name>coffee_2001_7f8__c5c5_0_1</lsp-name>
```


Linux Issues

- ➔ TCP over MPLS Encap Route unusably slow (~70kbyte/s)
  - ◆ On a route that made 1,5 Gbps with a non MPLS Route
- ➔ Interface TX drops
- ➔ Random chunks of segments missing
- ➔ Long story short: modprobe mpls_gso

State of Rollout

- Currently running on internal testing Machines
- Pending deployment of dedicated SDN Controller Machines
- Traffic Engineering Controller pending

Thank You!

Questions?