

gRIBI

A gRPC Service for RIB Injection

David Murray <dm@arista.com>

Introduction

- **gRIBI** is a **gRPC** service to **inject** entries into the **RIB**
- We will look at:
 - Existing approaches for route injection, their challenges and how gRIBI helps overcome them
 - Details about the gRIBI service
 - Walk through a simple weighted route injection scenario

Motivation

- Existing approaches* for route injection include
 - Direct programming of forwarding plane entries (P4Runtime, OpenFlow)
 - Use existing routing protocols to inject entries
 - >> e.g., BGP SR-TE Policy, BGP-LU for egress peer engineering.
 - Device APIs using a vendor SDK

* something, something ... I2RS

Motivation (contd.)

- Direct programming assumes
 - Controller(s) have full view of device's forwarding table.
 - Controller(s) can modify all hardware tables
 - Requires controller to know about resolving routes (usually IGP) and reacting to changes
 - adds complexity to overall system

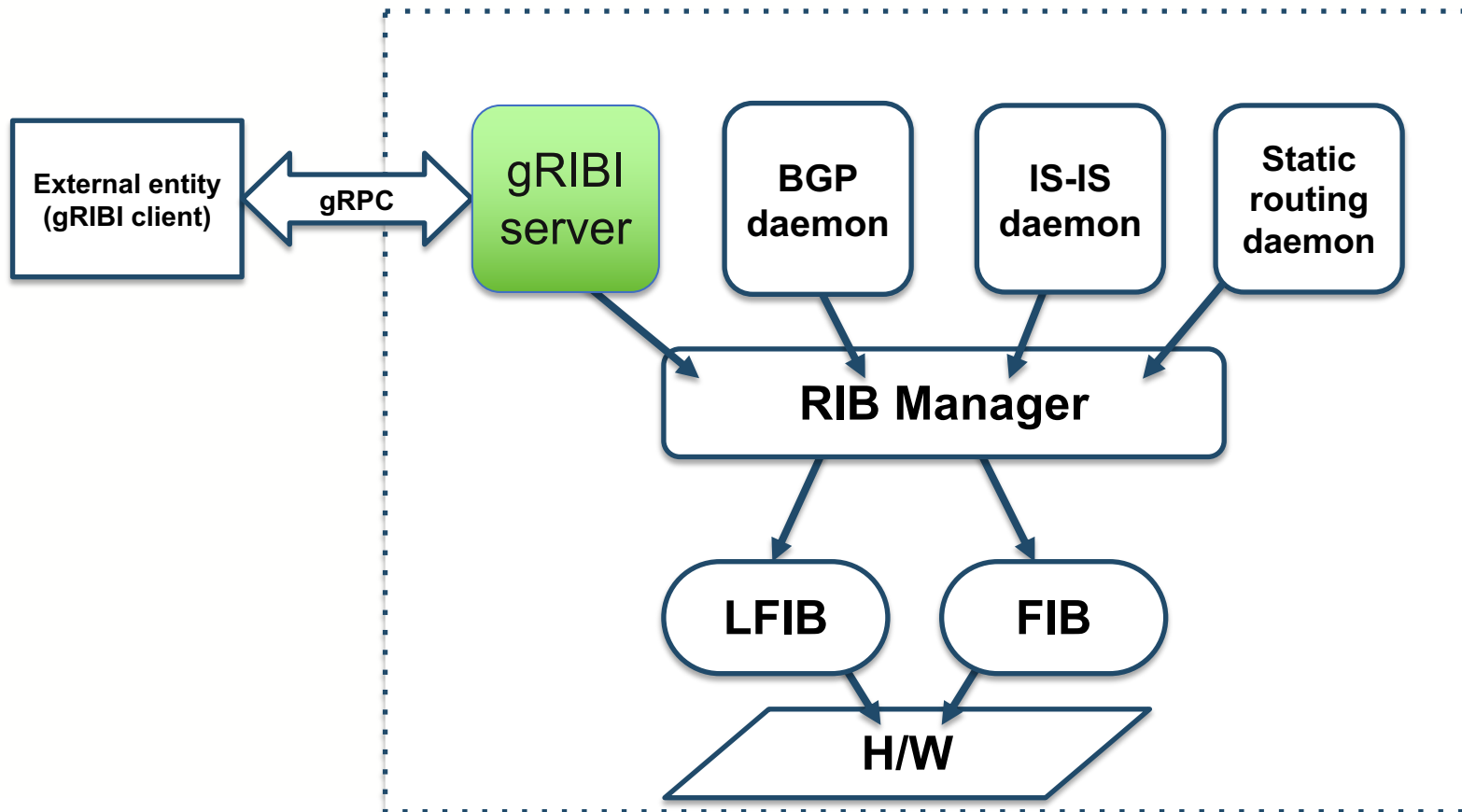
Motivation (contd.)

- Using a routing protocol involves:
 - Force fitting data model and routes to constraints of protocol (for example BGP NLRI uniqueness and affecting BGP best path Algo in the context of BGP SR-TE Policy)
 - No notion of transactional semantics
 - No acknowledgments of programming
- Using a device/vendor-specific API isn't open and portable

gRIBI

- gRPC service to inject (and query) routing table entries into a network device's RIB from an external entity (say a controller)
- From device's PoV, control plane service where injected entries are just another source to device's RIB(s)

gRIBI as a Control Plane Service



Transactional semantics for programming operation

- Every programming operation request from the external entity has an (unique) “id”
- Device responds with programming response for every request using the “id” which allows the external entity to tie back to a specific operation

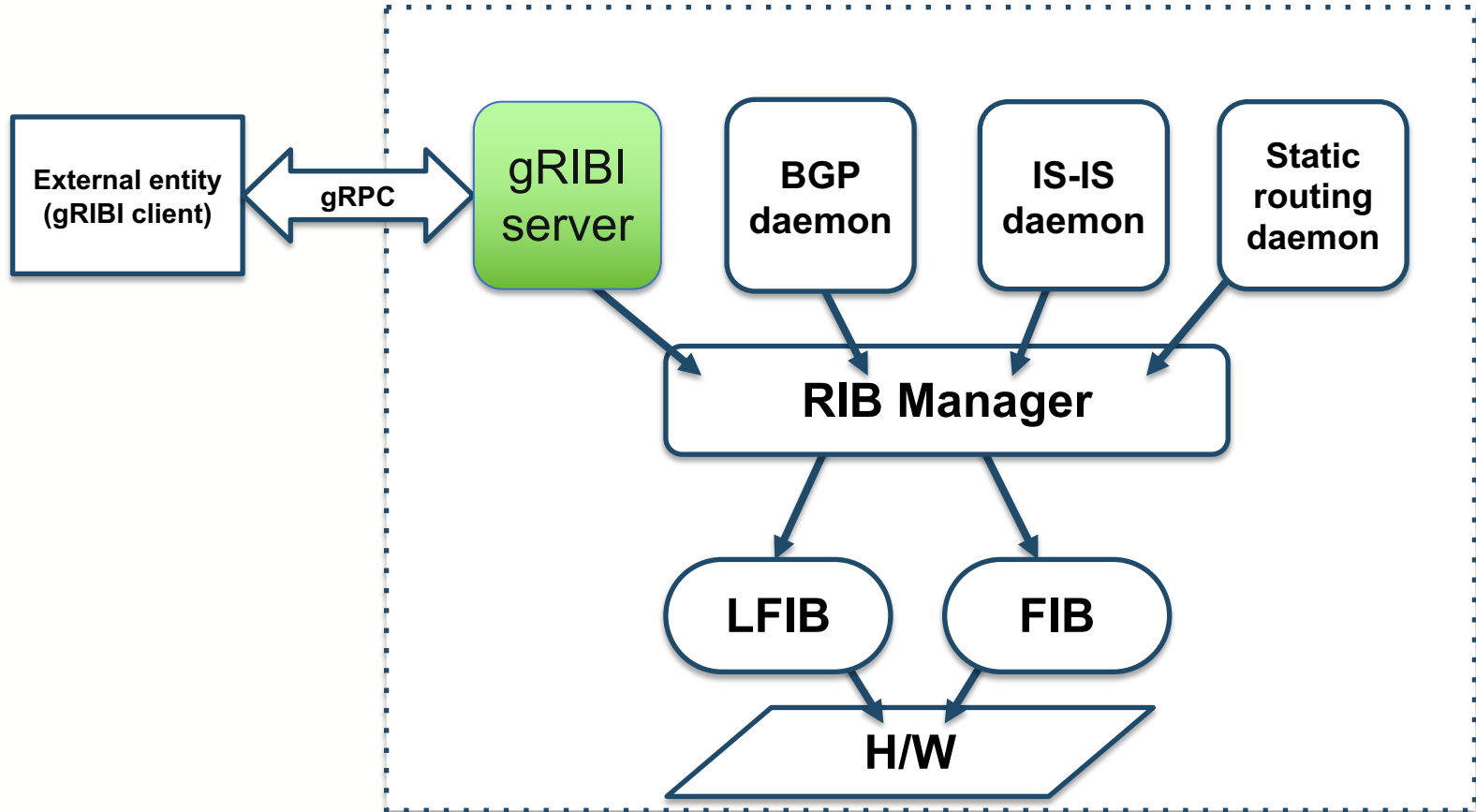
Other Features

- FIB programming ACK support
 - Acknowledgement from the device can separately indicate the status of the programming in the device's software RIB and hardware FIB
 - enables the controller to do something intelligent based on the response from the device
- Support for **redundant clients**
 - i.e., active/standby and active/active
- **Persistence** of programmed entries
 - Entries programmed by client persist in RIB and FIB on client disconnect and gRIBI daemon restart
- Support for gRPC **transport security** (mTLS/TLS/SPIFFE-ID)
 - Provides secure connections from external entity to device

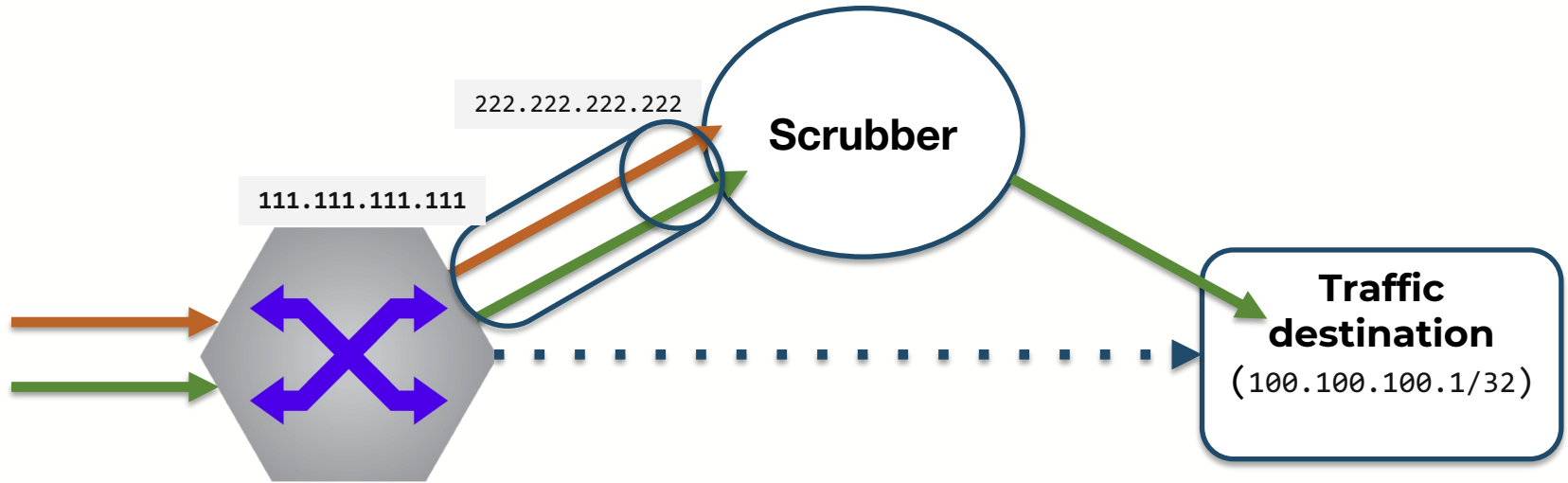
Example Applications

- Inject route entries into a VRF for scrubbing traffic for DDoS mitigation
 - gRIBI injected entry is another route with its own type and preference
 - Next hops are recursively resolved in the RIB like for any other route from a routing protocol
- Injecting a Labeled FIB entry that points to a WECMP set of label stacks akin to BSID steering in SR Policy
- Variations on these themes for selective tunnel-based traffic engineering

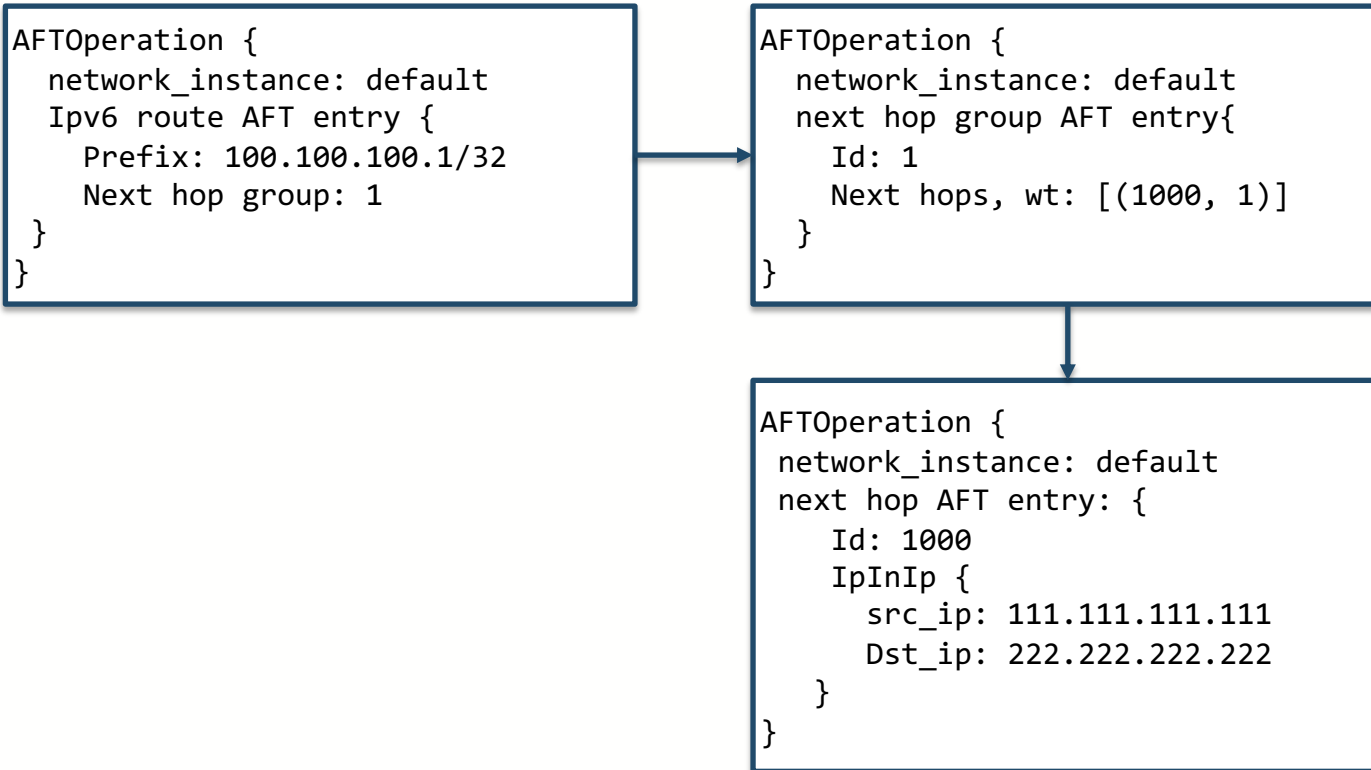
Route injection, *not* config



Traffic scrubbing for DDoS mitigation



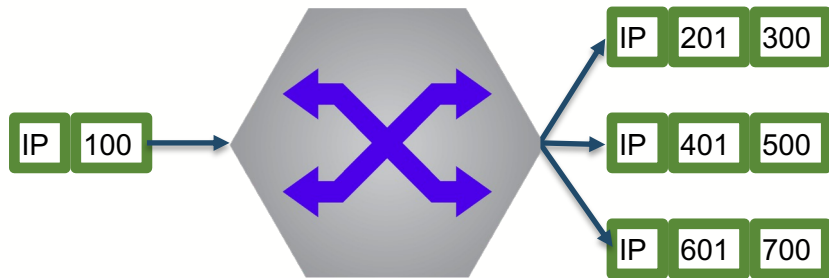
Example: prefix forwarding to IPinIP tunnel



Example: MPLS traffic to LSPs

```
AFTOperation {  
  network_instance: default  
  MPLS AFT entry {  
    Label: 100  
    Next hop group: 1  
  }  
}
```

```
AFTOperation {  
  network_instance: default  
  next hop group AFT {  
    Id: 1  
    Next hops, wt: [(1000, 1), (2000,2), (3000, 5)]  
  }  
}
```



```
AFTOperation {  
  network_instance: default  
  next hop AFT:  
    Id: 1000  
    Pushed MPLS label: 201  
    Pushed MPLS label: 300  
  }  
}  
  
{ ... Id: 2000}  
{ ... Id: 3000}
```

* not all WECMP legs are shown

RPCs

- **Modify**
 - Inject entries, client parameters
- **Get**
 - Retrieve entries with RIB/FIB installation state
- **Flush**
 - OOB delete all entries

Example

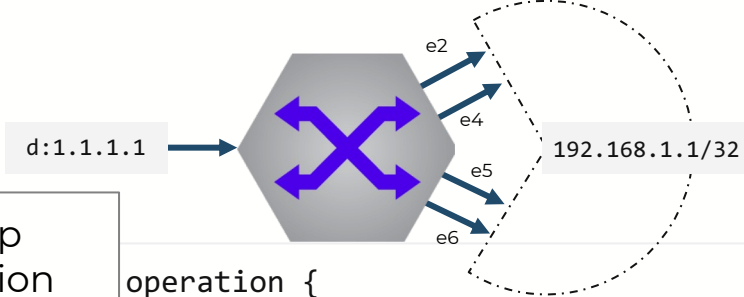
connection
params

```
params {  
  redundancy:  
  SINGLE_PRIMARY  
  persistence: PRESERVE  
}  
  
election_id {  
  low: 1  
}  
  
{...}
```

next-hop
construction

```
operation {  
  id: 3  
  network_instance: "default"  
  op: ADD  
  next_hop {  
    index: 3  
    next_hop {  
      ip_address {  
        value: "192.168.1.1"  
      }  
      interface_ref {  
        interface {  
          value: "Ethernet5"  
        }  
      }  
    }  
  }  
  election_id {  
    low: 1  
  }  
}
```

d:1.1.1.1



```
operation {  
  id: 4  
  network_instance: "default"  
  op: ADD  
  next_hop {  
    index:  
    next_hop {  
      ip_address {  
        value: "192.168.1.1"  
      }  
      interface_ref {  
        interface {  
          value: "Ethernet2"  
        }  
      }  
    }  
  }  
  election_id {  
    low: 1  
  }  
}
```

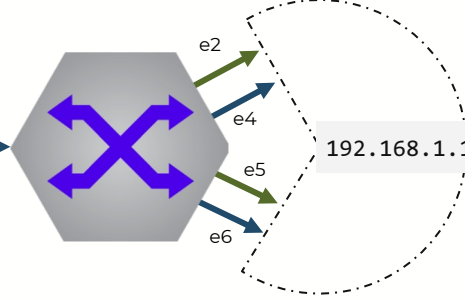

Example

```
{ ... }  
operation {  
  id: 8  
  network_instance: "default"  
  op: ADD  
  next_hop_group {  
    id: 2  
    next_hop_group {  
      next_hop {  
        index: 3  
        next_hop {  
          weight {  
            value: 1  
          }  
        }  
      }  
    }  
  }  
}
```

next-hop group
construction

```
next_hop {  
  index: [REDACTED]  
  next_hop {  
    weight {  
      value: [REDACTED]  
    }  
  }  
}  
}  
election_id {  
  low: 1  
}
```

d:1.1.1.1



route
association

```
{ ... }  
operation {  
  id: 12  
  network_instance: "default"  
  op: ADD  
  ipv4 {  
    prefix: "1.1.1.1/32"  
    ipv4_entry {  
      next_hop_group_network_instance {  
        value: "default"  
      }  
      next_hop_group {  
        value: 2  
      }  
    }  
  }  
  election_id {  
    low: 1  
  }  
}
```

Example

network-instance →

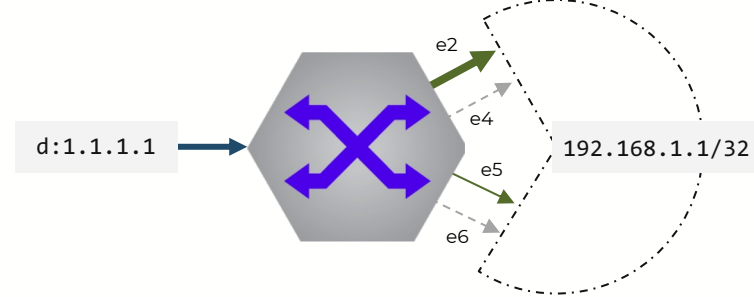
```
aip1#show ip route
show ip route
VRF: default
WARNING: Some of the routes are not programmed in
kernel, and they are marked with '%'.
Codes: C - connected, S - static, K - kernel,
O - OSPF, IA - OSPF inter area, E1 - OSPF external type 1,
E2 - OSPF external type 2, N1 - OSPF NSSA external type 1,
N2 - OSPF NSSA external type2, B - Other BGP Routes,
B I - iBGP, B E - eBGP, R - RIP, I L1 - IS-IS level 1,
I L2 - IS-IS level 2, O3 - OSPFv3, A B - BGP Aggregate,
A O - OSPF Summary, NG - Nexthop Group Static Route,
V - VXLAN Control Service, M - Martian,
DH - DHCP client installed default route,
DP - Dynamic Policy Route, L - VRF Leaked,
G - gRIBI, RC - Route Cache Route
```

dynamically
programmed
entries for 1.1.1/32 →

```
Gateway of last resort is not set
G% 1.1.1.1/32 [5/0] via 192.168.1.1, Ethernet2, weight 3/4
via 192.168.1.1, Ethernet5, weight 1/4
G% 2.2.2.2/32 [5/0] via 192.168.1.1, Ethernet4, weight 3/4
via 192.168.1.1, Ethernet6, weight 1/4
C 3.3.3.0/24 is directly connected, Ethernet5
C 4.4.4.0/24 is directly connected, Ethernet2
C 5.5.5.0/24 is directly connected, Ethernet6
C 6.6.6.0/24 is directly connected, Ethernet4
C 10.30.1.0/24 is directly connected, Ethernet1
C 10.40.1.0/24 is directly connected, Ethernet3
S 192.168.1.1/32 [1/0] is directly connected, Ethernet2
is directly connected, Ethernet4
is directly connected, Ethernet5
is directly connected, Ethernet6
C 192.168.201.4/30 is directly connected, Ethernet4
```

dynamically
programmed
next-hop weights

static routes for
192.168.1.1 recursive
resolution



Conclusions

- gRIBI provides a new and *open* mechanism for programming network device RIB state
- Supports a range of forwarding paradigms
 - IP tunnels, surgical routing, VRF population, etc.
 - not constrained to classic traffic engineering technologies (e.g. RSVP)
- Multiple implementations do exist
- Operators can utilize modern tools and software engineering techniques to interact with the RIB and customize forwarding behaviors
 - <https://github.com/openconfig/gribi>
 - <https://github.com/openconfig/gribigo>

Thank You

arista.com

Backup Slides