gRIBI
an open interface for programming your RIB

11-may, 2023

steve ulrich (sulrich@arista.com)

overview

gRIBI is a gRPC service to inject entries into a RIB

- why?
- use cases
- gRIBI service overview
- sample interaction

why?

- existing approaches* for route injection
    - 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

* mumble something … I2RS

why (not do direct programming of the device)?

- existing approaches* for route injection
  - **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

direct programming assumes …

- controller(s) have full view of device's forwarding table.
- controller(s) can modify all hardware tables
- a controller must know about resolving routes (usually IGP) and react to changes
- adds complexity to the overall system

why (not use a routing protocol)?

- existing approaches* for route injection
  - 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

using a routing protocol involves …

- force fitting data model and routes to constraints of protocol
  - e.g.: BGP NLRI uniqueness and affecting BGP best path Algo in the context of BGP SR-TE Policy
- no transactional semantics
- no programming acks

why? (not use a vendor SDK)

- existing approaches* for route injection
  - 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**

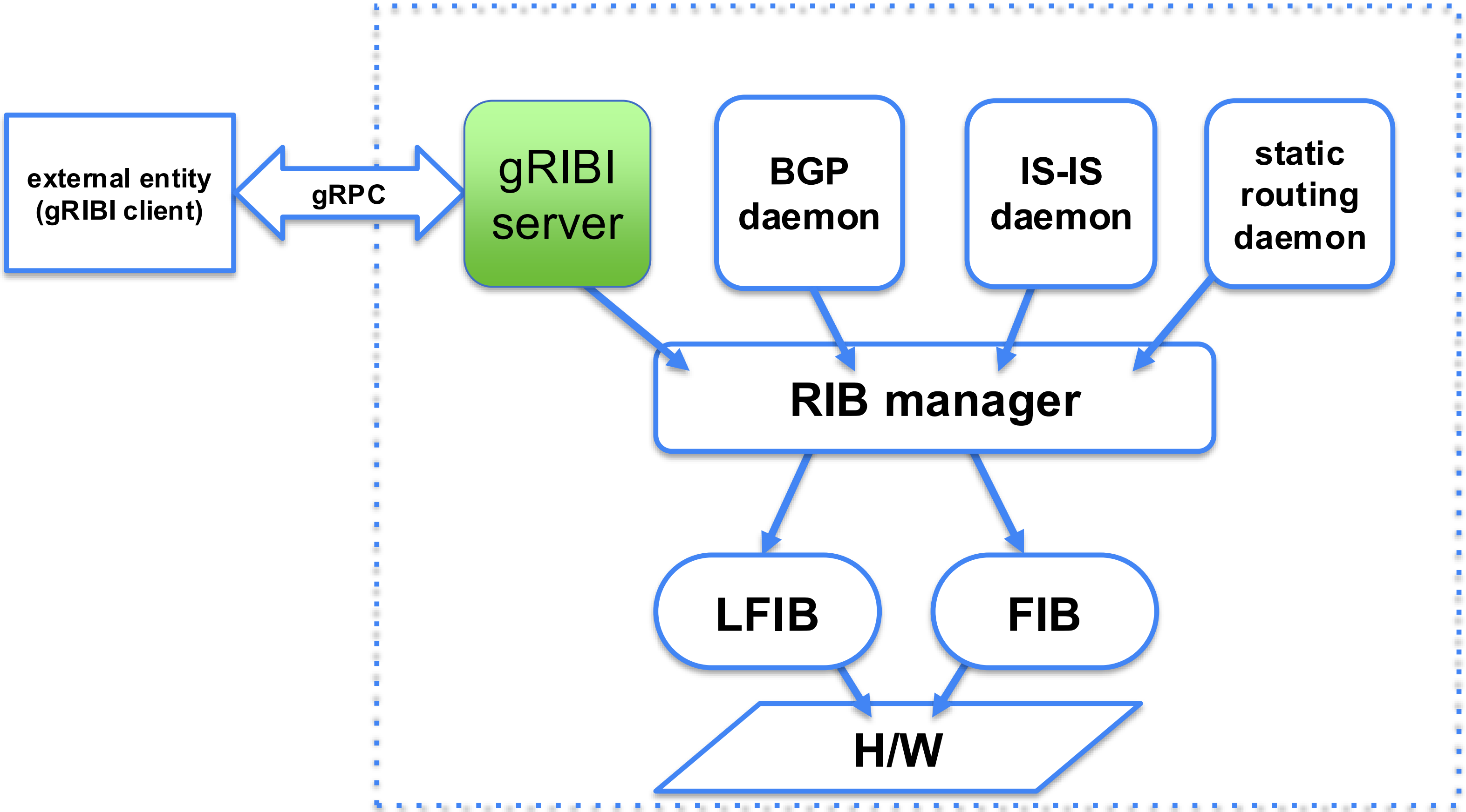
using a vendor SDK …

- this 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 the network device's PoV - control plane service where injected entries are just another source to the device RIB(s)

gRIBI: as a control plane service



gRIBI data model

the data model is the existing OpenConfig Abstract Forwarding Table (AFT) converted to protobuf

afts	container
ipv4-unicast	container
ipv4-entry[prefix]	list
prefix	leaf
state	container
prefix	leaf
counters	container
entry-metadata	leaf
origin-protocol	leaf
decapsulate-header	leaf
oc-aftni:next-hop-group	leaf
oc-aftni:next-hop-group-network-instance	leaf
oc-aftni:origin-network-instance	leaf

gRIBI features

transactional semantics for programming operation

- each programming operation request from the client has an (unique) “id”
- network device responds with programming response for every request using the “id” which allows the client to tie things back to a specific operation

FIB programming ACK

- 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

gRIBI features (cont'd)

support for redundant clients

- i.e., active/standby and active/active

programmed entry persistence

- client programmed entries persist in RIB & FIB on client disconnect and gRIBI daemon restart

modern and developer friendly RPC transport

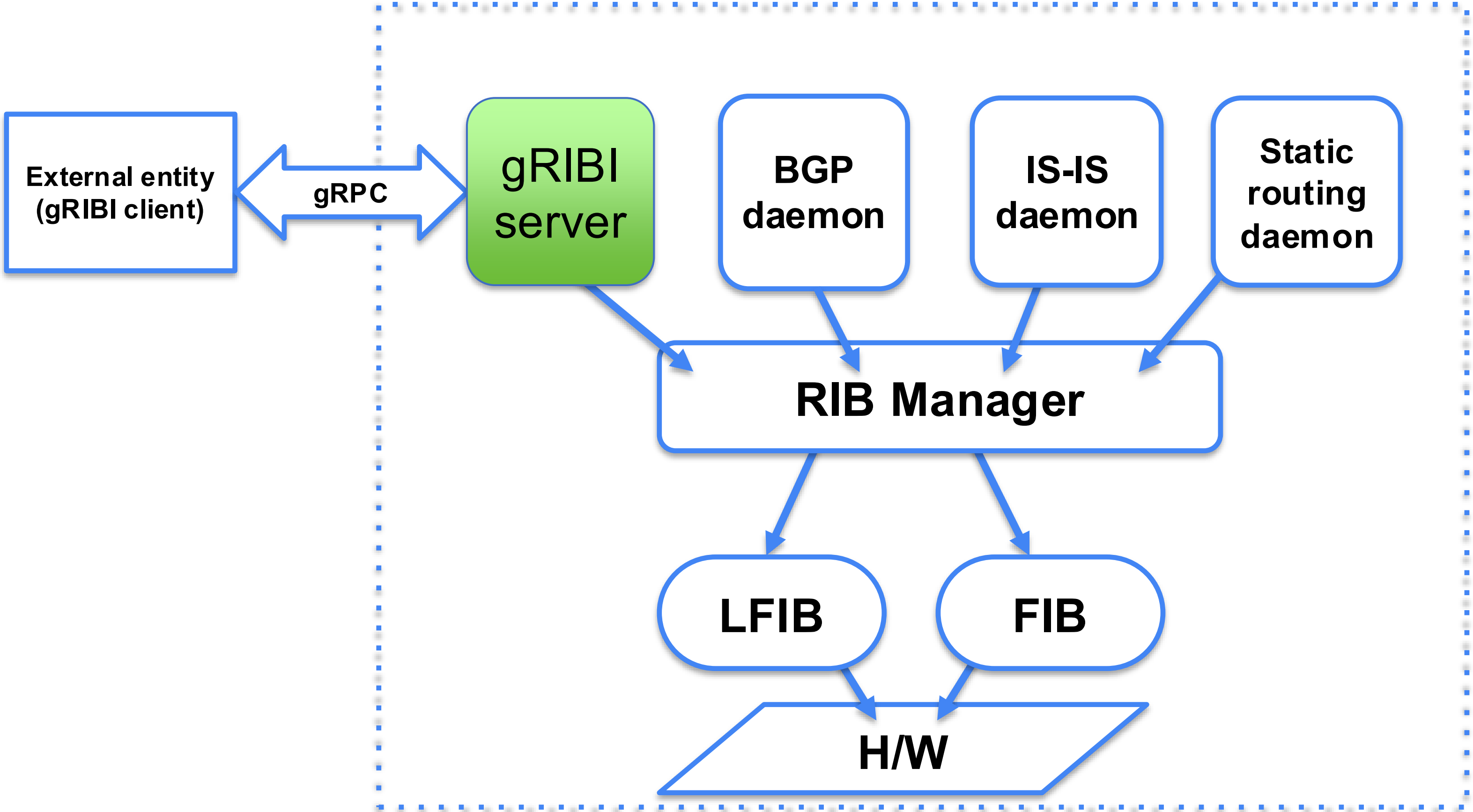
- leverages support for gRPC transport security (mTLS/TLS/SPIFFE-ID) to provide secure connections from client to device

use cases

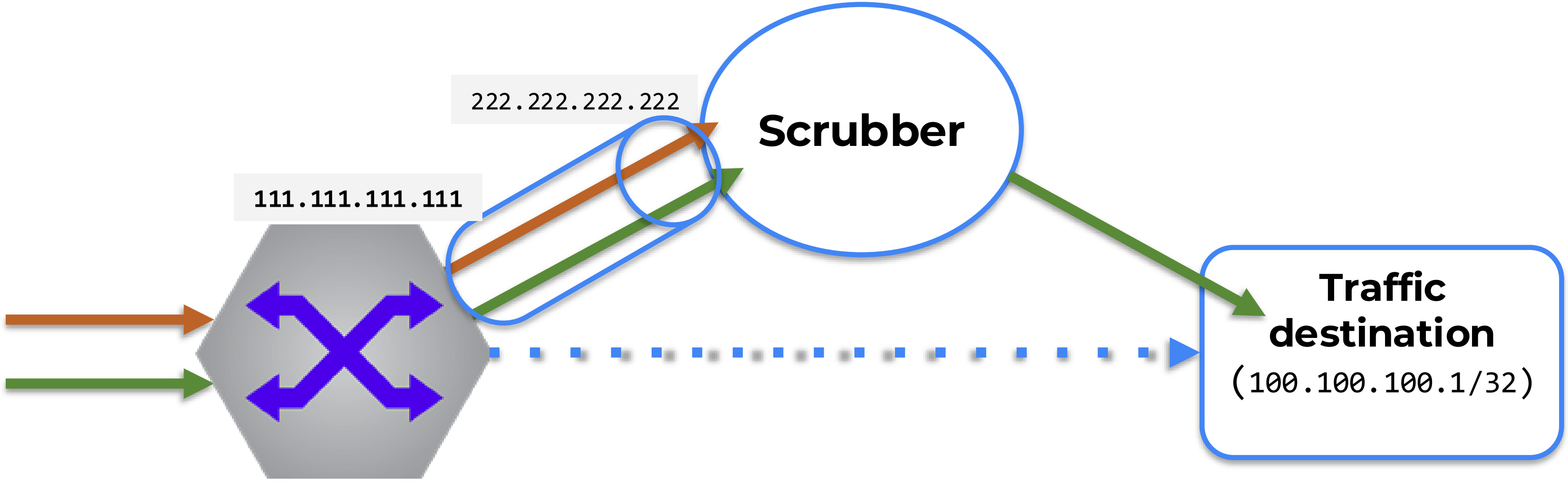
example applications / use cases

- 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



DDoS scrubbing



DDoS scrubbing: prefix forwarding into IPinIP tunnel

```
AFTOperation {
  network_instance: default
  Ipv6 route AFT entry {
    Prefix: 100.100.100.1/32
    Next hop group: 1
 }
}
```

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

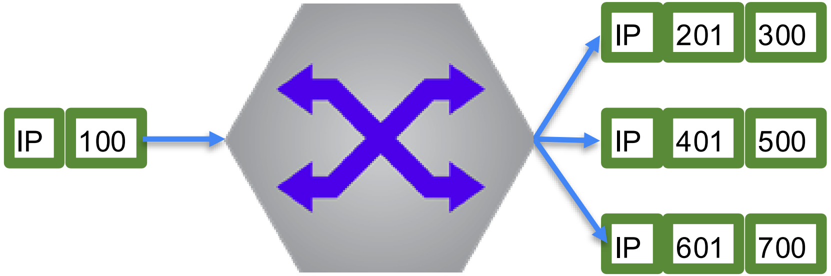
```
AFTOperation {
 network_instance: default
 next hop AFT entry: {
    Id: 1000
    IpInIp {
      src_ip: 111.111.111.111
      Dst_ip: 222.222.222.222
   }
}
```

example: MPLS traffic to LSPs (SRTE w/ACKs!)

```
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`
- `Get`
- `Flush`

`Modify` - inject RIB entries & negotiate client parameters

```
rpc Modify(stream ModifyRequest) returns (stream ModifyResponse)
```

each `ModifyRequest` AFT operation has:

- id
- network instance (VRF)
- operation (add/replace/delete)
- entry

each `ModifyResponse` has:

- id
- RIB, FIB Status
- timestamp

`Modify` - session parameters

upon client connection it sends session parameters in a `ModifyRequest` to specify the type of connection and desired behaviors

- client redundancy
  - active/active, active/standby
- AFT persistence
  - persist or delete
- ACK type
  - RIB ACK or RIB+FIB ACK

`Modify` - election ID

- used by device to determine active client
- when a client connects, it sends its election ID
- device responds with highest election ID it knows about
- each AFT Operation also has the election ID and the gRIBI server only processes operations from the client with the highest election ID

`Get` - fetch device state

```
rpc Get(GetRequest) returns (stream GetResponse)
```

- a `GetRequest` from the client can request all AFT entries from all network-instances (aka VRFs) or filter on a VRF and/or AFT type
- the router streams entries along with last RIB and FIB acknowledgement status

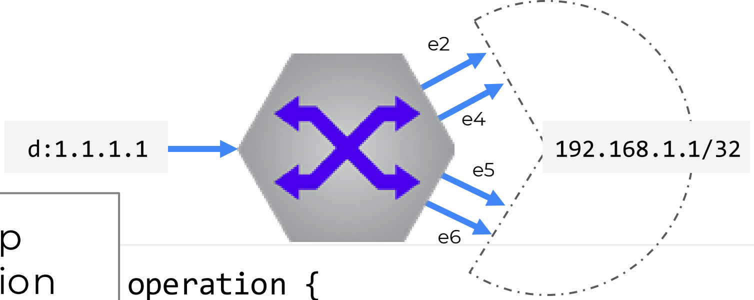
`Flush` - clear one or all VRFs

```
rpc Flush(FlushRequest) returns (FlushResponse)
```

- `FlushRequest` contains
  - election ID (or an override to ignore election ID)
  - a network-instance (VRF) name or all network-instances (VRFs)
- `FlushResponse` contains a result and timestamp.
- meant to be used by external entity during controller malfunction.

example interaction

UCMP example



connection params

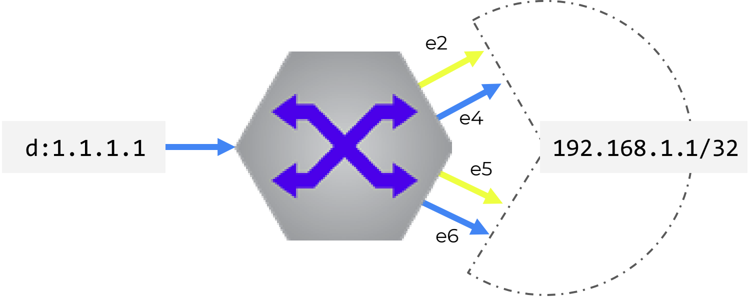
```
params {
  redundancy:
SINGLE_PRIMARY
  persistence: PRESERVE
}

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
  }
}
```

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

UCMP example



next-hop group construction

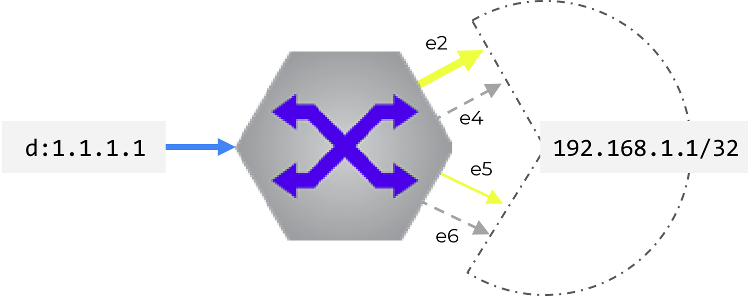
```
{ ... }

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 {
        index: 4
        next_hop {
          weight {
            value: 3
          }
        }
      }
    }
  }
  election_id {
    low: 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
  }
}
```

UCMP 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

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 entries for 1.1.1.1/32

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 (RSVP)
- multiple implementations do exist
- as an industry we're reaching a point where operators can start to utilize modern tools and software engineering techniques to interact with the RIB and customize forwarding behaviors

questions?

references

- gRIBI - Github repository
  - Motivation document
  - Specification
  - Protobuf definitions
- gRIBIGo Reference implementation

thank you