xBGP: When You Can't Wait for the IETF and Vendors

Thomas Wirtgen, Quentin De Coninck, Randy Bush, Laurent Vanbever and **Olivier Bonaventure**

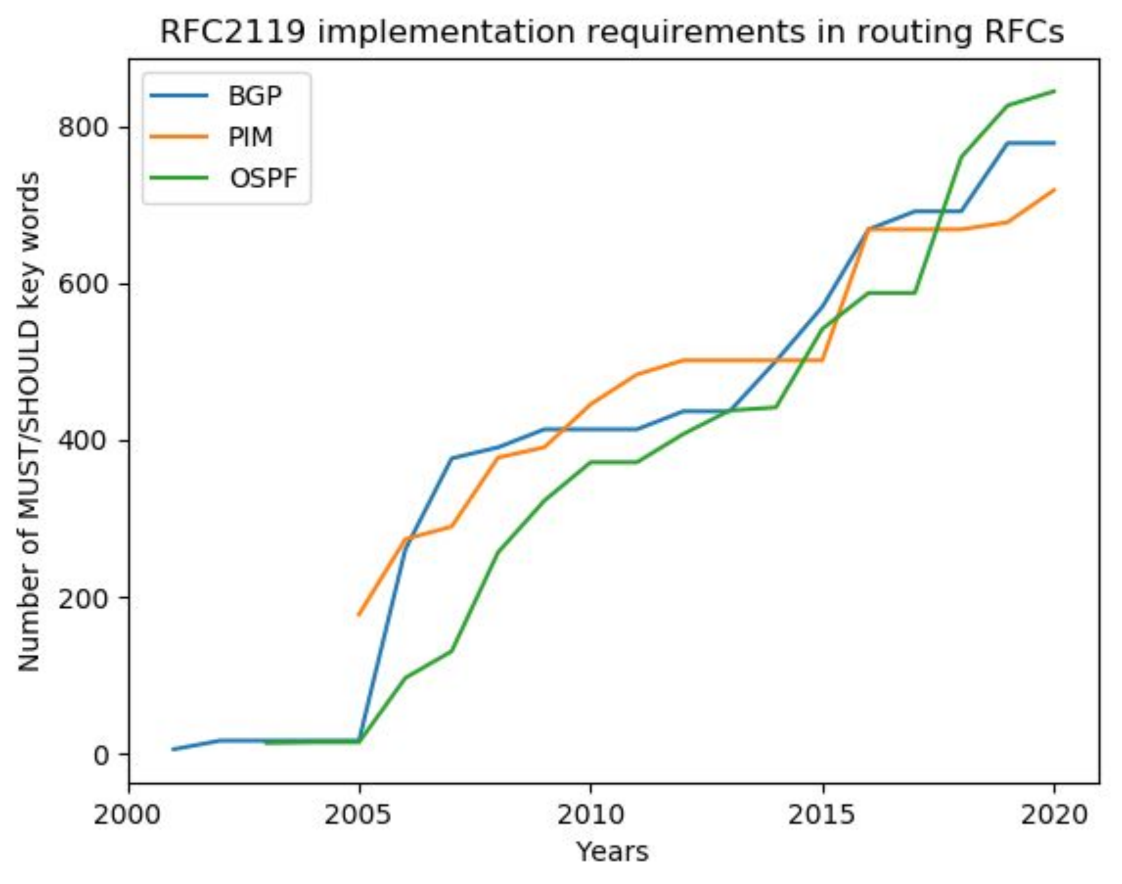
UCLouvain icteam

ETH zürich

Networked Systems
ETH Zürich — seit 2015

IIJ
Internet Initiative Japan

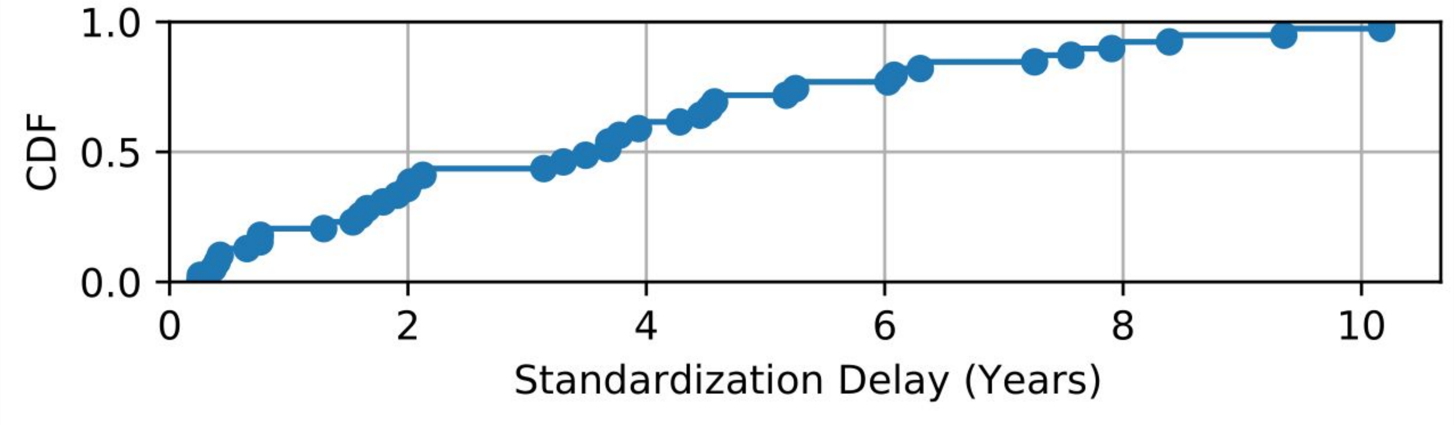
Routing protocols evolve regularly to address new requirements from operators



Problem #1: Networks evolve, as do routing protocols

The evolution is complex:

1. Standardization by the IETF (3.5 years in average for BGP)
2. Implementation on the vendor OS
3. Update routers of networks



Problem #2: Large networks use diverse routers

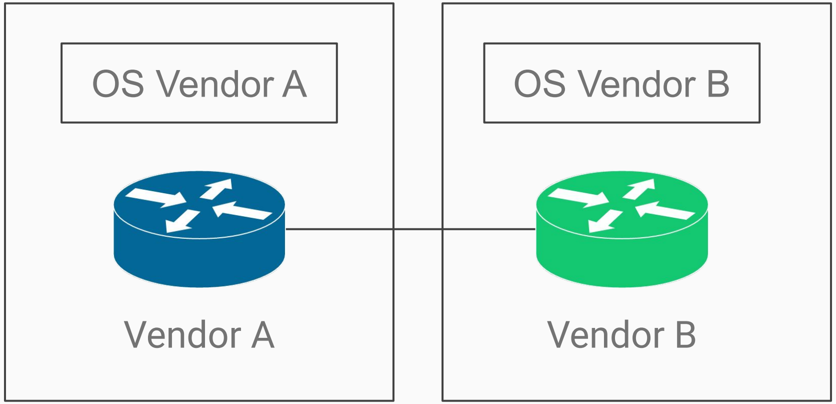
Vendors do not propose the same set of extensions on their routers

The configuration of these routers differs as well

```
routing-options {
  router-id 1.1.1.1;
  autonomous-system 65001;
}

protocols {
  bgp {
    group Session-to-R1 {
      type external;
      neighbor 1.1.1.2 {
      peer-as 65002;
    }
  }
}
```

Simple Juniper configuration file




```
router bgp 65001
  bgp router-id 1.1.1.1
  neighbor 1.1.1.2 remote-as 65002
```

Simple Cisco configuration file

How do we answer requests for protocol extensions ?

```
IDR Working Group                                         R. Raszuk, Ed.
Internet-Draft                                              Bloomberg LP
Intended status: Standards Track                                R. White
Expires: December 2, 2016                                       Ericsson
                                                                 J. Dong
                                                     Huawei Technologies
                                                            May 31, 2016
```

BGP Path Record Attribute
draft-raszuk-idr-bgp-pr-05

Abstract

The BGP protocol contains number of built in mechanisms which records
information about the routers
of reachability information c
are chosen by the protocol.
and ORIGINATOR_ID attributes
permanent routing loops are n
particular destination. Howe
other useful information alon
through which reachability in
helpful to the operator in or
the BGP control plane.

Table of Contents

1. Introduction 3
2. Protocol Extensions 3
  2.1. BGP Path Record Attribute 3
  2.2. BGP Per Hop TLV 4
    2.2.1. Host Name sub-TLV 6
    2.2.2. Time Stamp sub-TLV 6
    2.2.3. Next hop record sub-TLV 6
    2.2.4. Path count sub-TLV 7
    2.2.5. Origin Validation sub-TLV 7
    2.2.6. Geo-location sub-TLV 7
    2.2.7. BGP System Load sub-TLV 8

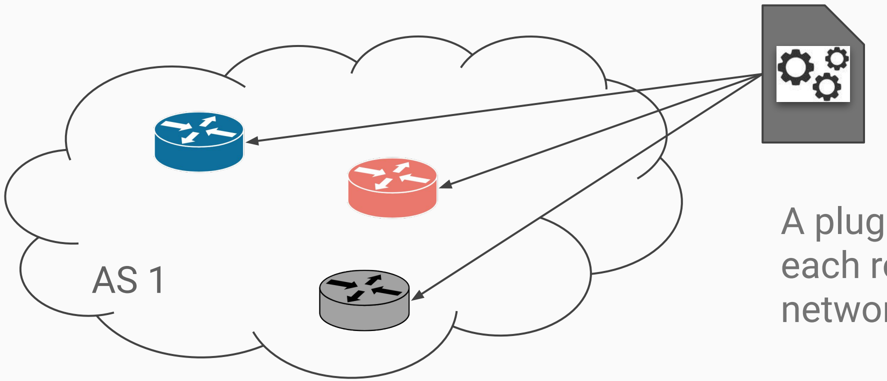
Agenda

- **xBGP: a Paradigm Shift**
- Adding a new feature with xBGP
- Uses Cases

xBGP: a paradigm shift

Each xBGP compliant router exposes a simple API that allows to dynamically extend the protocol with platform-independent code that we call plugins.

Network operators can program their routers directly using plugins.

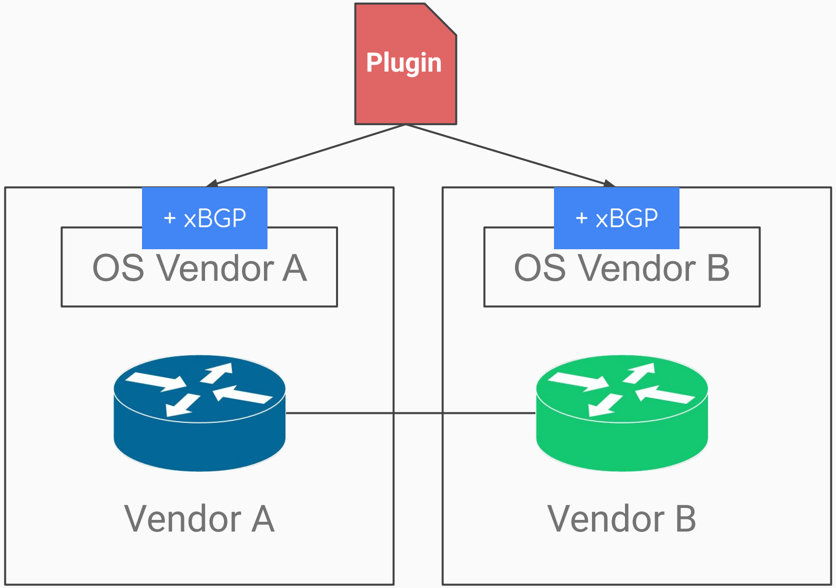


A plugin is injected for each router of the network

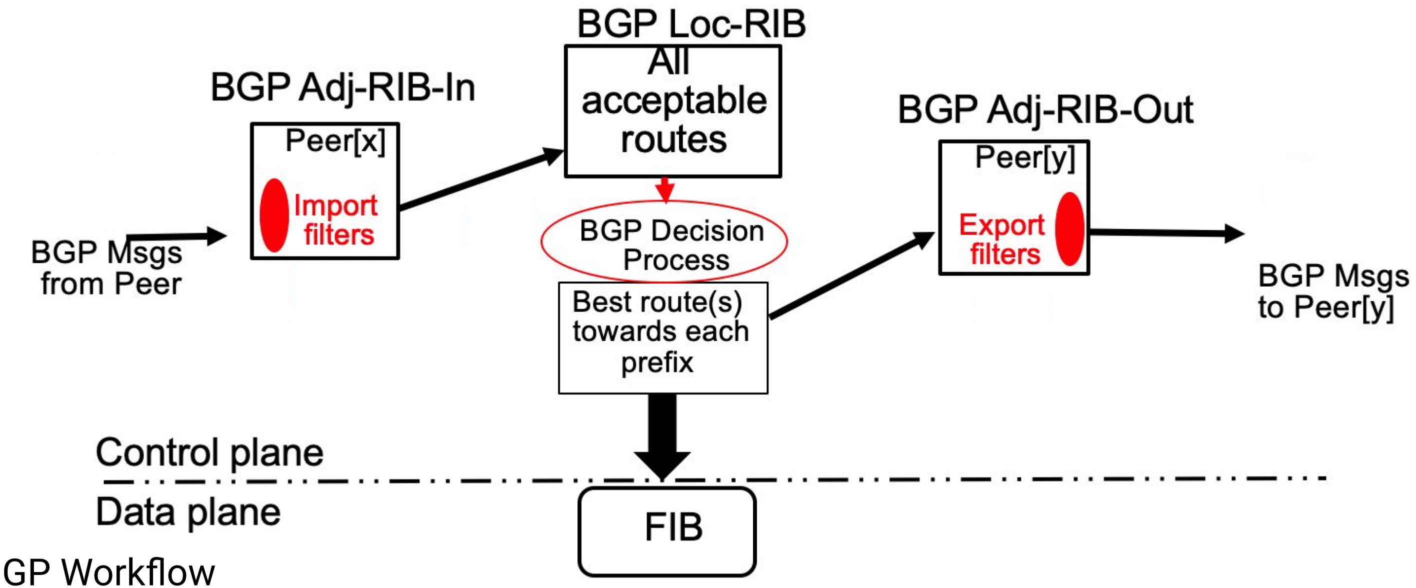
All xBGP routers expose the same API

Each router adds xBGP on top of its implementation

With xBGP, routers expose a common API.

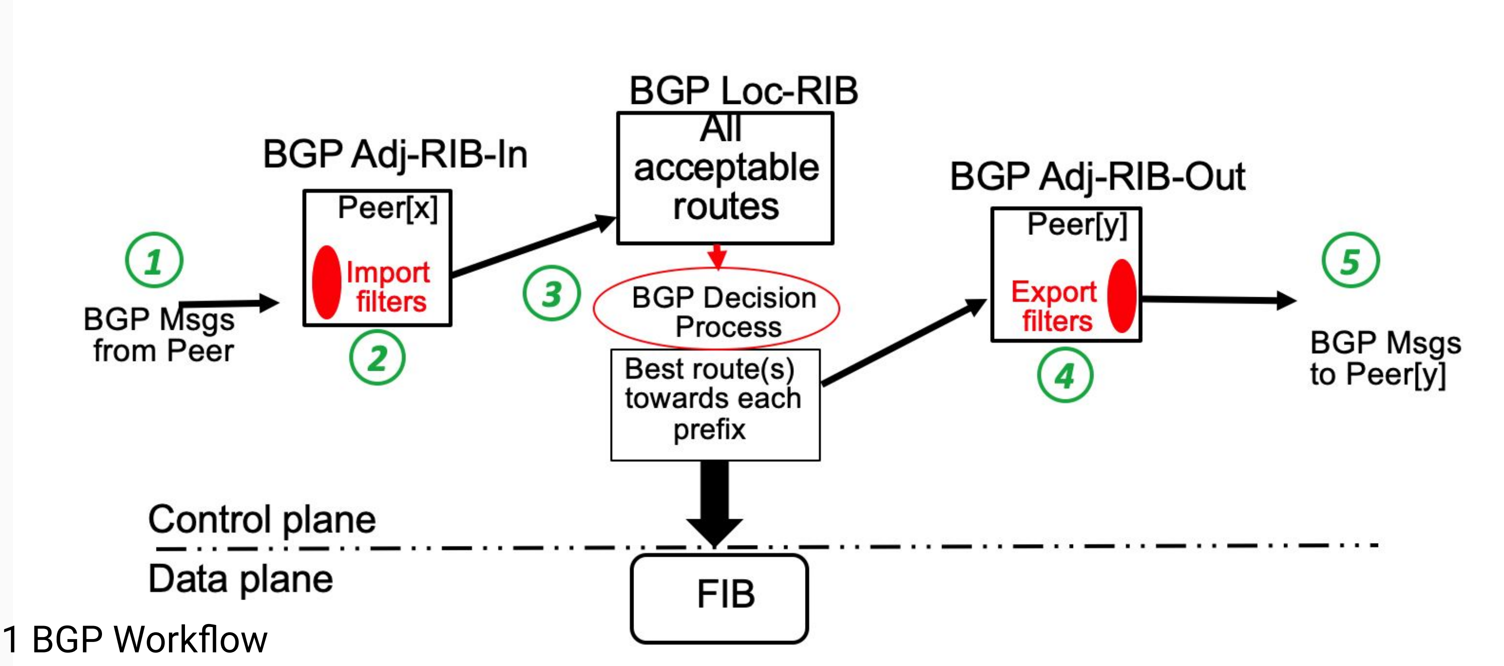


BGP workflow



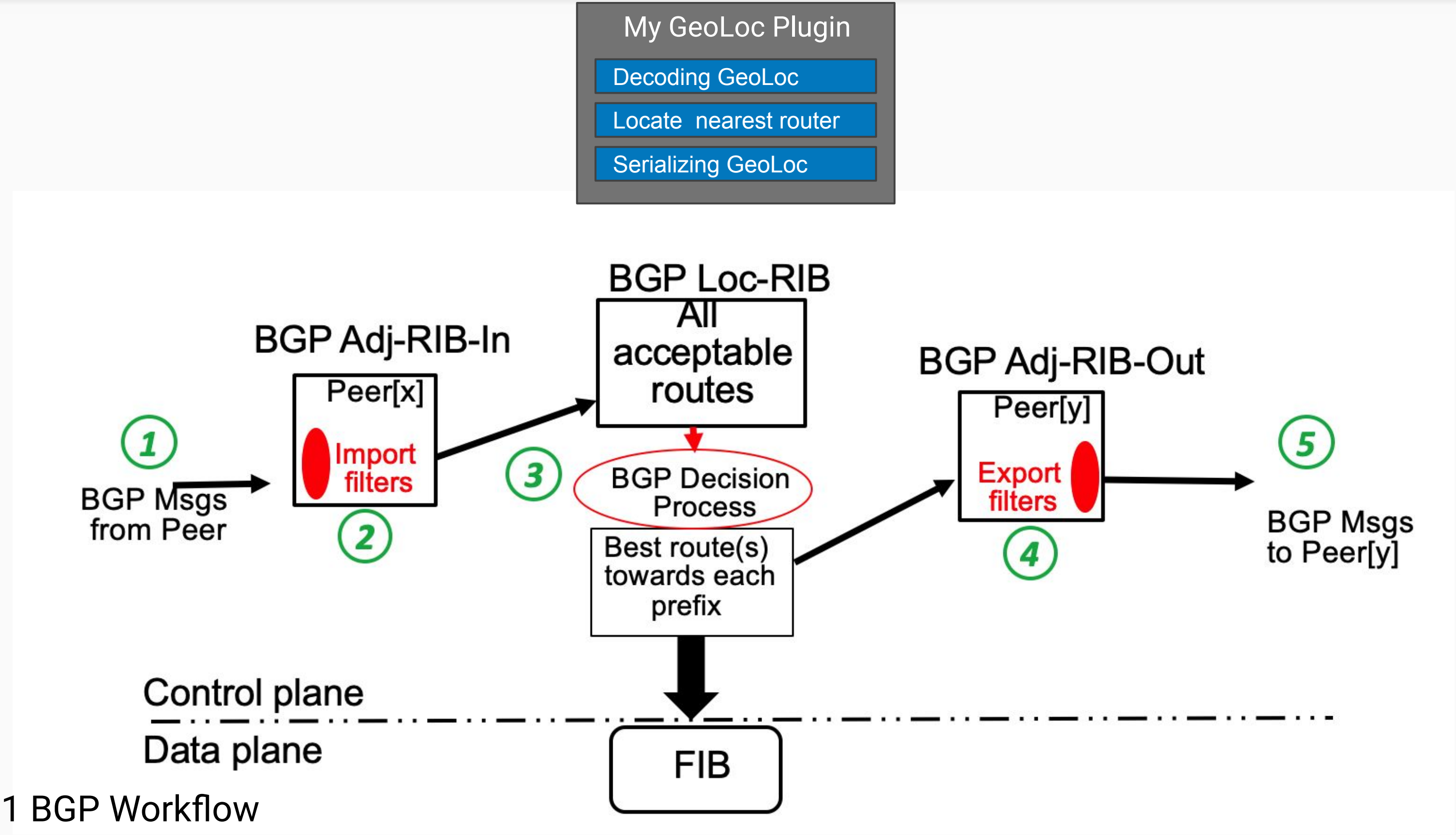
RFC 4271 BGP Workflow

BGP workflow



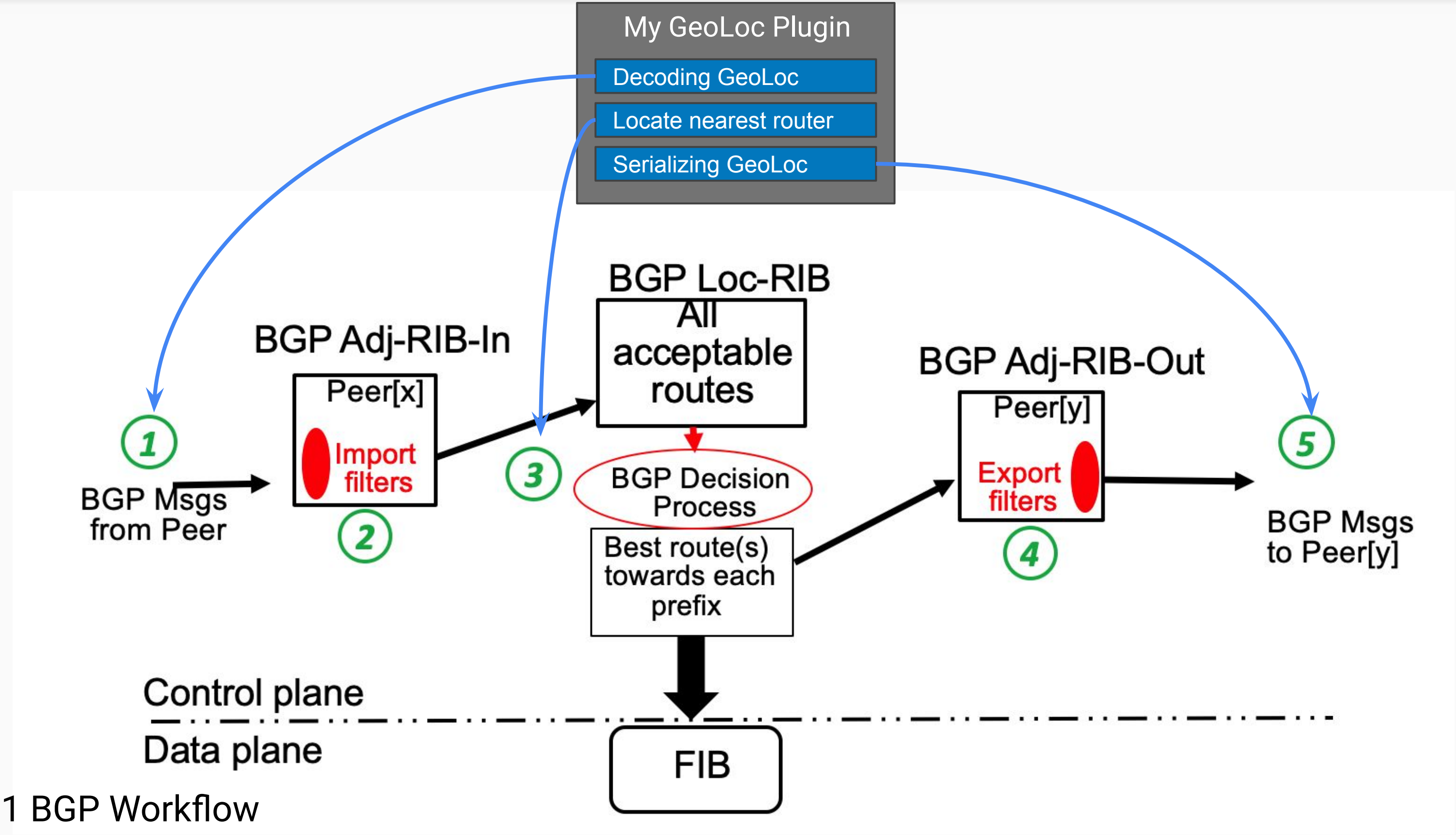
RFC 4271 BGP Workflow

A plugin to support a geoloc attribute



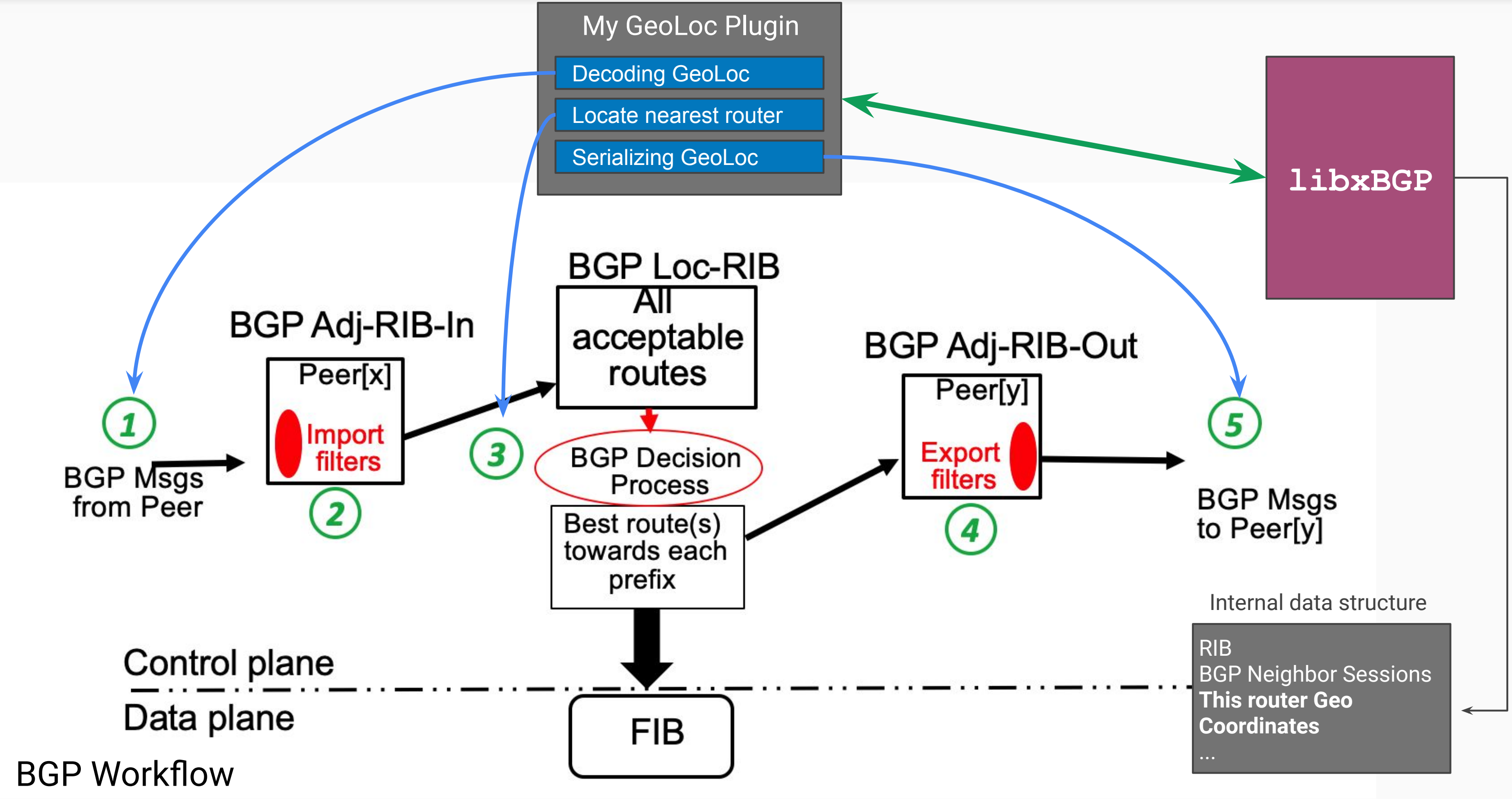
RFC 4271 BGP Workflow

A plugin to support a geoloc attribute



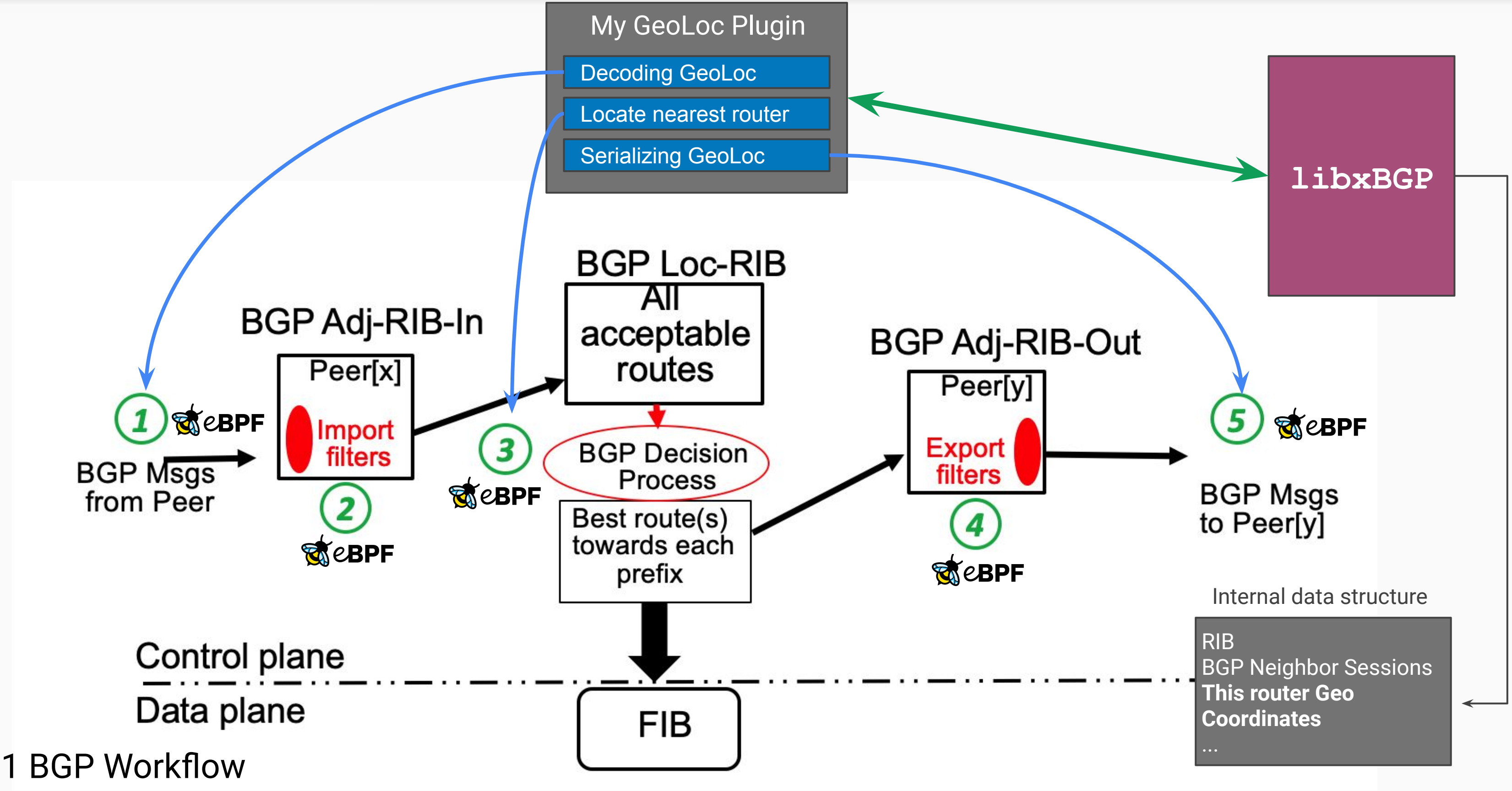
RFC 4271 BGP Workflow

A plugin to support a geoloc attribute



RFC 4271 BGP Workflow

A plugin to support a geoloc attribute



RFC 4271 BGP Workflow

Agenda

- xBGP: a Paradigm Shift
- Adding a new feature with xBGP
- **Uses Cases**

Implementation effort for xBGP

xBGP requires a little adaptation on the host BGP implementation

We have adapted both FRRouting and BIRD to be xBGP compliant

	FRRouting (LoC)	BIRD Routing (LoC)
Modification to the codebase	30	10
Insertion Points	73	66
Plugin API	624	415
`libxbgp`	3004 + dependencies	
User Space eBPF VM	2776	

Use Cases

1. Re-implementation of route reflectors (295 LoC)
2. Expressive filters
    - Route Origin Validation (126 LoC)
    - Valley Free path check for datacenters (81 LoC)
3. GeoLoc attribute (261 LoC)

Conclusion

With xBGP, BGP implementations can become truly extensible

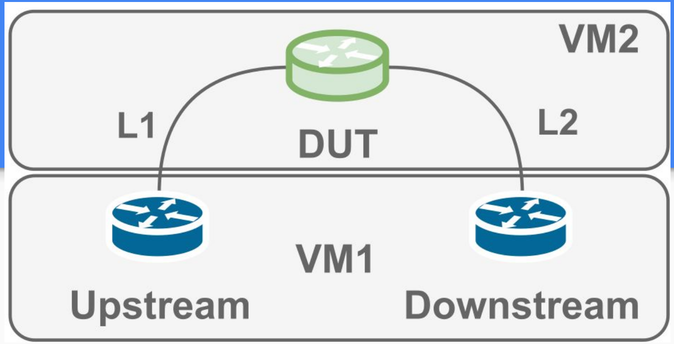
T. Wirtgen, Q. De Coninck, L. Vanbever, R. Bush, O. Bonaventure, *xBGP: When You Can't Wait for the IETF and Vendors*, Hotnets'20, Nov. 2020
See https://www.pluginized-protocols.org/xbgp for running source code

Next steps

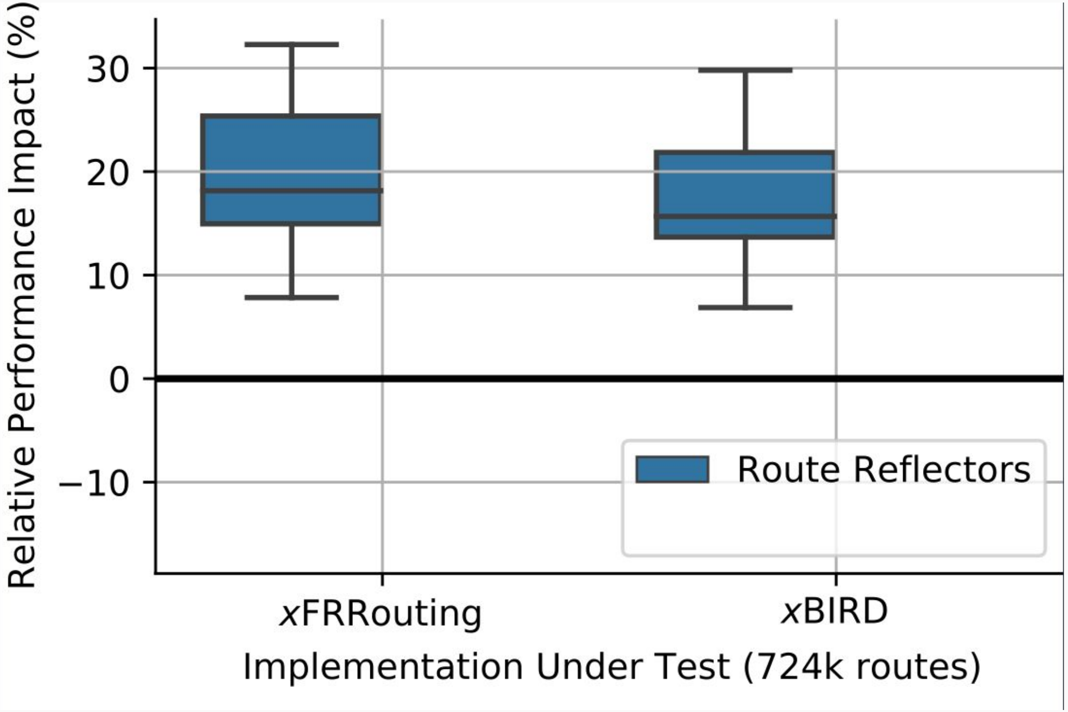
- Discuss with network operators to address other requirements
- Discuss with BGP implementors and IETF to precisely define the xBGP API
- Extend the approach to other routing protocols

Backup slides

Comparison with native code

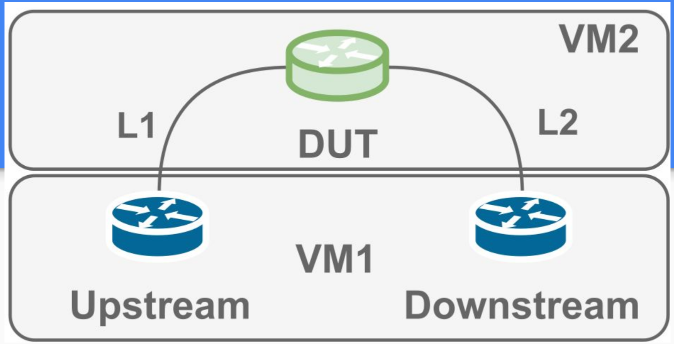


$$\frac{\text{Time xBGP - Time Native}}{\text{Time Native}} \times 100$$

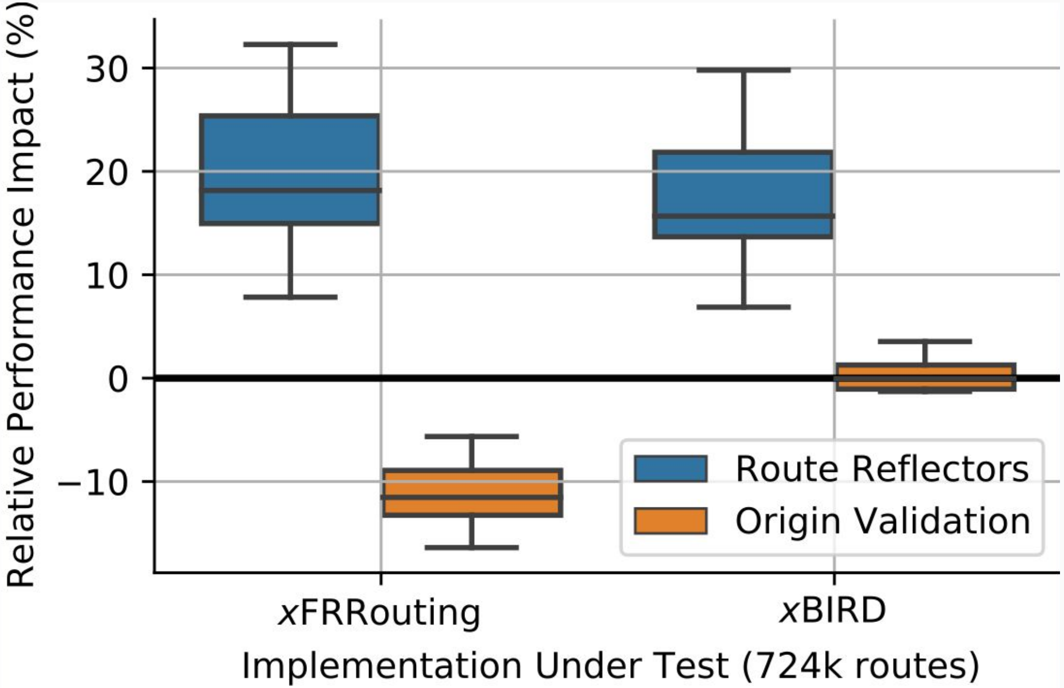


Lower is better

Comparison with native code

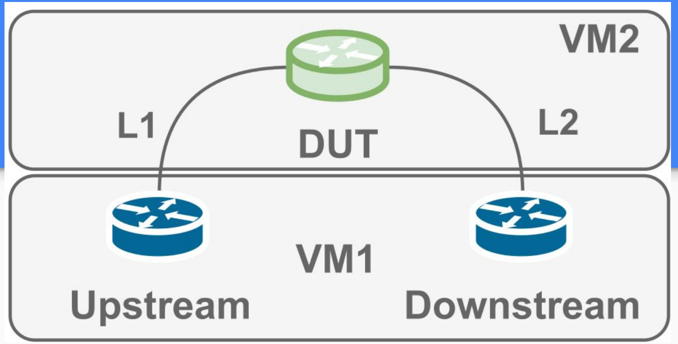


$$\frac{\text{Time xBGP - Time Native}}{\text{Time Native}} \times 100$$

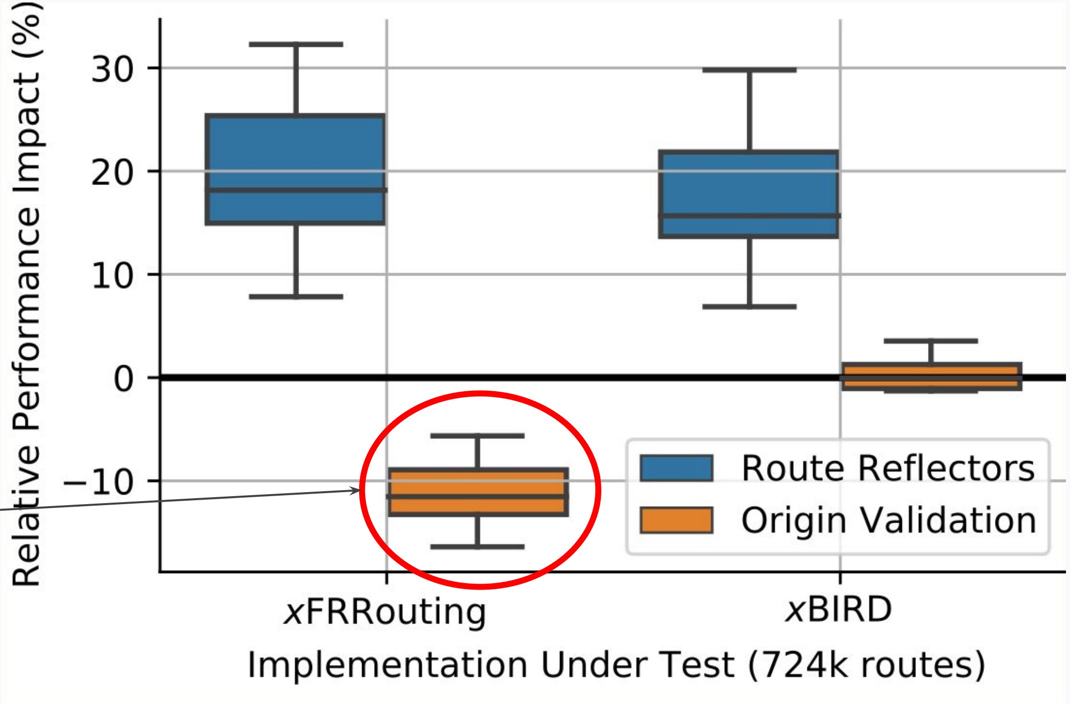


Lower is better

Comparison with native code



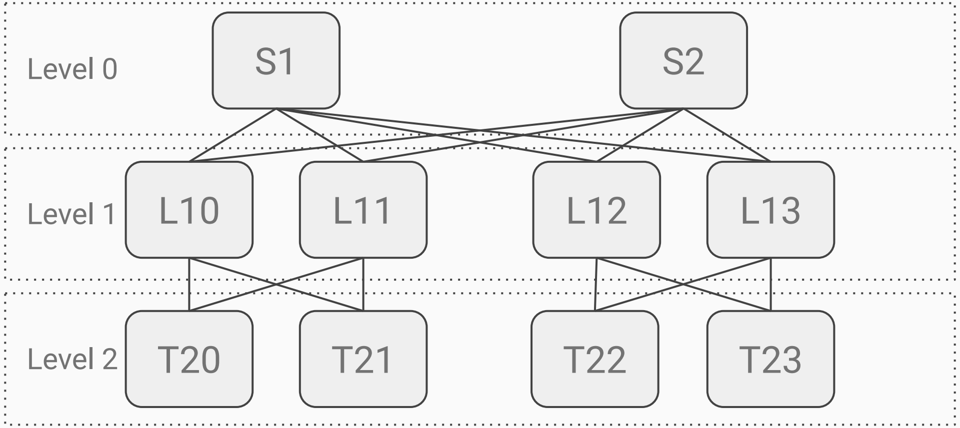
$$\frac{\text{Time xBGP - Time Native}}{\text{Time Native}} \times 100$$



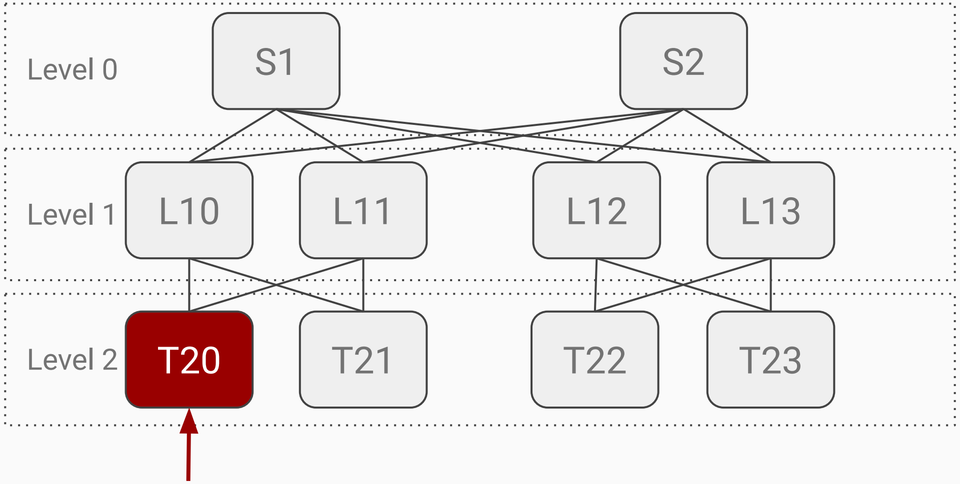
Native code uses a slower data structure

Lower is better

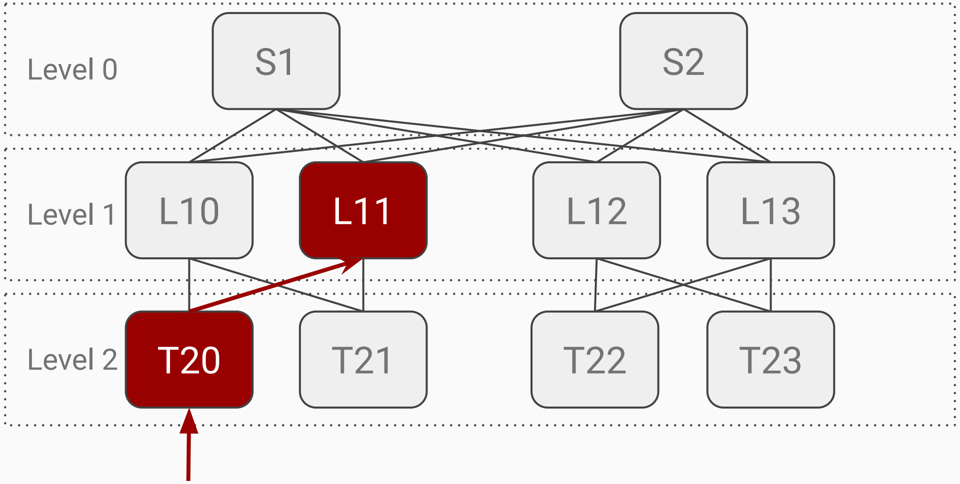
Valley Free path check



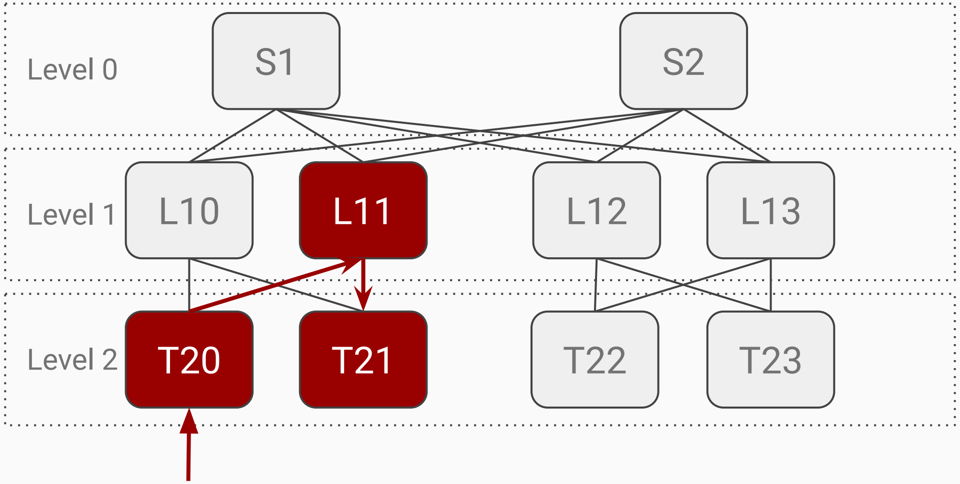
Valley Free path check



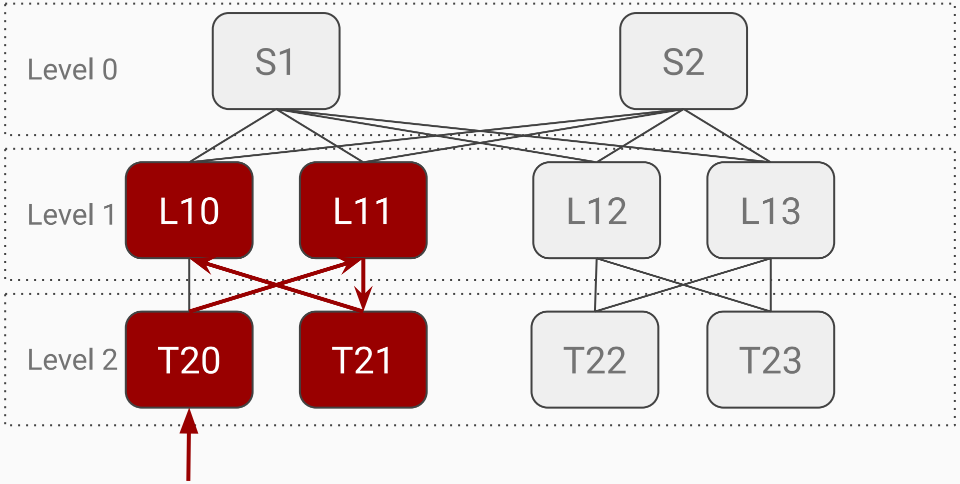
Valley Free path check



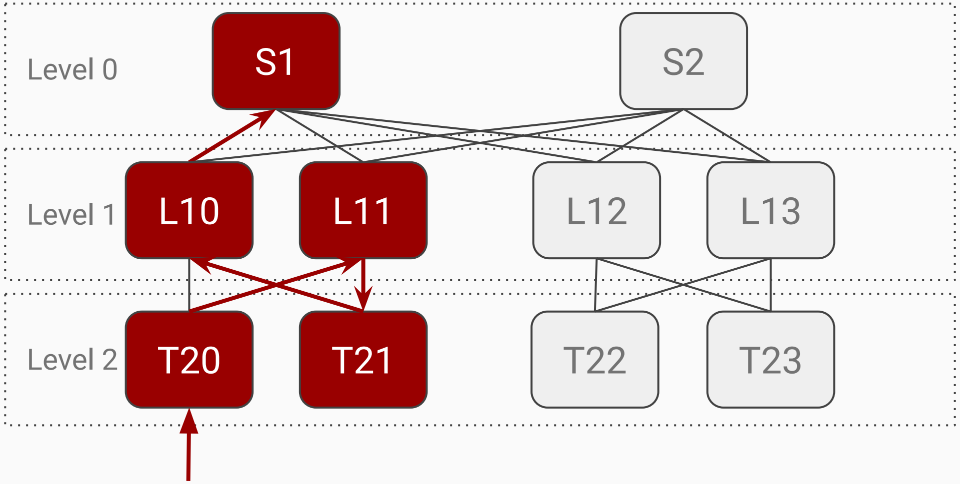
Valley Free path check



Valley Free path check



Valley Free path check



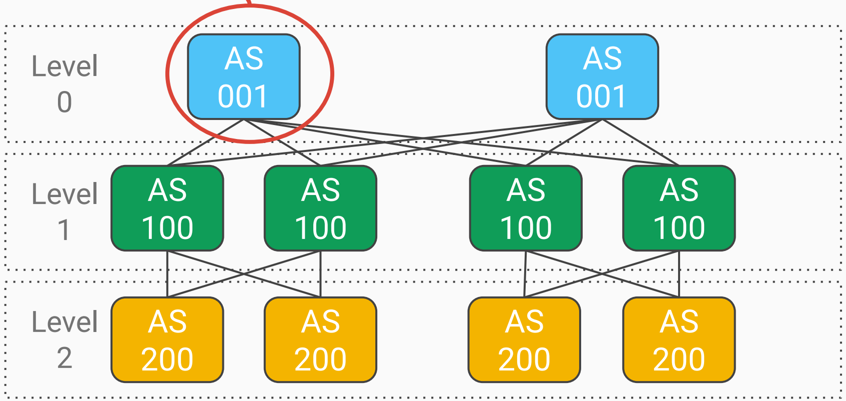
Valley Free path check

RFC7938 Use of BGP for Routing in Large-Scale Data Centers

MyRouterCli > show ip bgp

BGP Routing table information for VRF default
Router identifier 192.168.254.5, local AS number 1

	Network	Next Hop	Metric	LocPref	Weight	Path
* >Ec	192.168.10.0/24	192.168.255.20	0	100	0	100 200 i
* ec	192.168.10.0/24	192.168.255.4	0	100	0	100 200 i
* >Ec	192.168.254.3/32	192.168.255.4	1	100	0	100 200 i
* ec	192.168.254.3/32	192.168.255.20	0	100	0	100 200 i
* >Ec	192.168.254.4/32	192.168.255.20	0	100	0	100 200 i



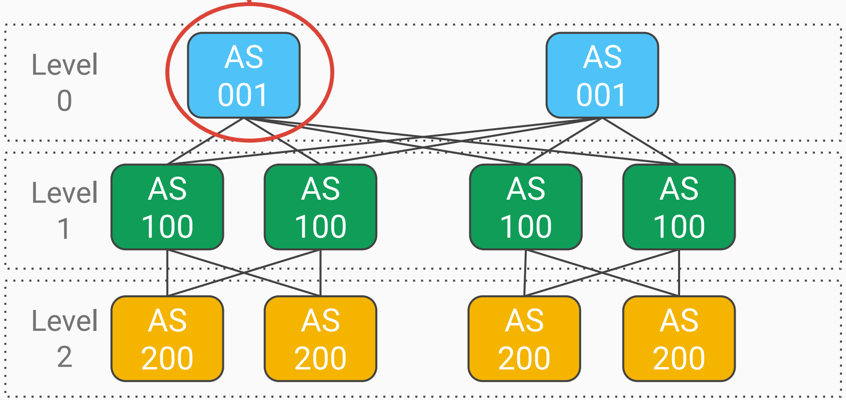
Valley Free path check

RFC7938 Use of BGP for Routing in Large-Scale Data Centers

MyRouterCli > **show ip bgp**

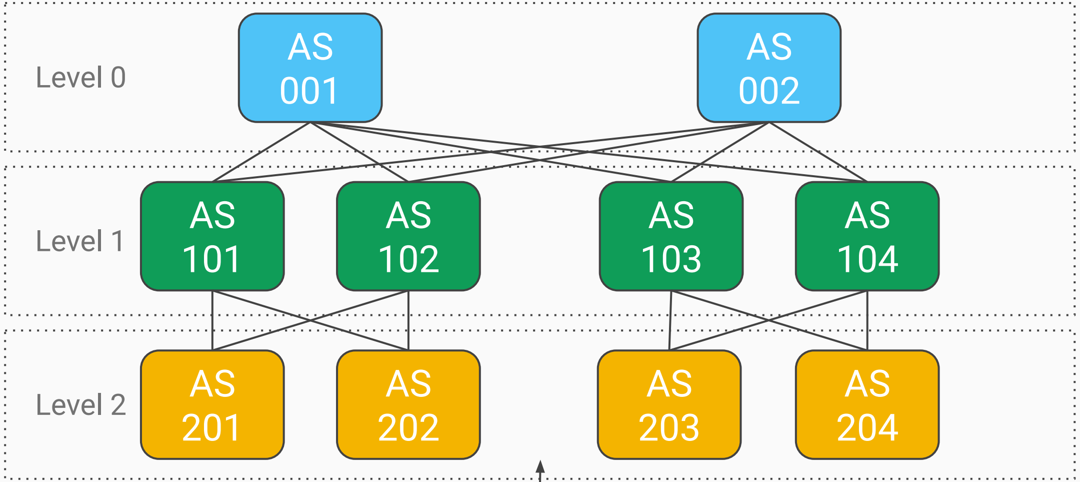
BGP Routing table information for VRF default
Router identifier 192.168.254.5, local AS number 1

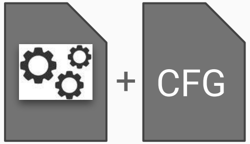
	Network	Next Hop	Metric	LocPref	Weight	Path
* >Ec	192.168.10.0/24	192.168.255.20	0	100	0	100 200 i
* ec	192.168.10.0/24	192.168.255.4	0	100	0	100 200 i
* >Ec	192.168.254.3/32	192.168.255.4	1	100	0	100 200 i
* ec	192.168.254.3/32	192.168.255.20	0	100	0	100 200 i
* >Ec	192.168.254.4/32	192.168.255.20	0	100	0	100 200 i



Where are these routes sourced from ?

Valley Free path check with xBGP





(81 LoC)

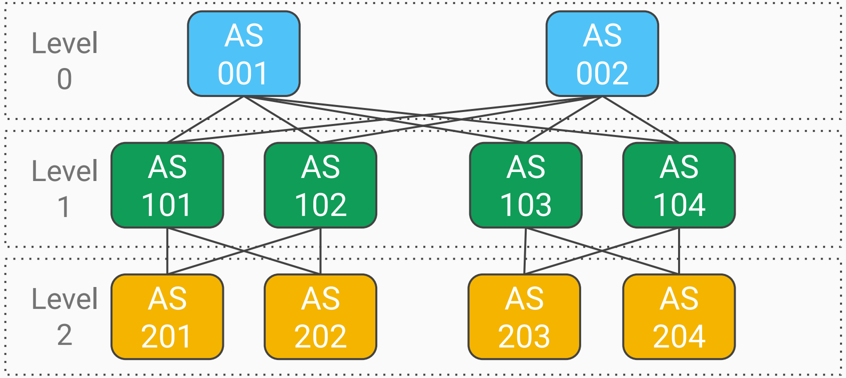
One plugin + one topology manifest for all routers !

Valley Free path check with xBGP

```c
uint64_t valley_free_check(args_t *args UNUSED) {
 /* variable declaration omitted  */
 attr = get_attr_from_code(AS_PATH_ATTR_CODE);
 peer = get_src_peer_info();
 if (!attr || !peer) return FAIL;

 my_as = peer->local_bgp_session->as;
 as_path = attr->data;
 as_path_len = attr->len;

 while (i < as_path_len) {
   i++; /* omit segment type  */
   segment_length = as_path[i++];
   for (j = 0; j < segment_length - 1; j++) {
     curr_as = get_u32(as_path + i);
     i += 4;
     if (!valley_check(next_as, curr_as)) return PLUGIN_FILTER_REJECT;
   }
 }
 next();
 return FAIL;
}
```



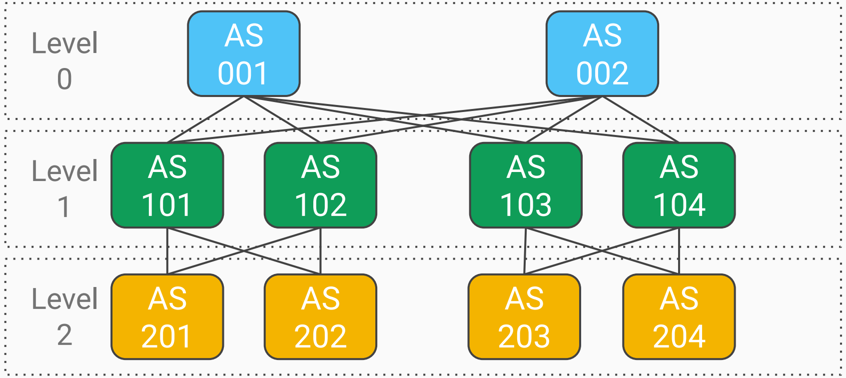
Valley Free path check with xBGP

```
uint64_t valley_free_check(args_t *args UNUSED) {
 /* variable declaration omitted  */
 attr = get_attr_from_code(AS_PATH_ATTR_CODE);
 peer = get_src_peer_info();
 if (!attr || !peer) return FAIL;

 my_as = peer->local_bgp_session->as;
 as_path = attr->data;
 as_path_len = attr->len;

 while (i < as_path_len) {
   i++; /* omit segment type  */
   segment_length = as_path[i++];
   for (j = 0; j < segment_length - 1; j++) {
     curr_as = get_u32(as_path + i);
     i += 4;
     if (!valley_check(next_as, curr_as)) return PLUGIN_FILTER_REJECT;
   }
 }
 next();
 return FAIL;
}
```

Retrieve data from the host implementation



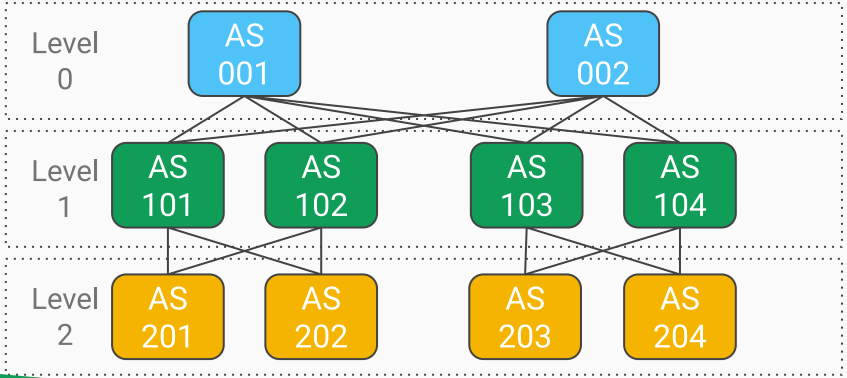
Valley Free path check with xBGP

```c
uint64_t valley_free_check(args_t *args UNUSED) {
 /* variable declaration omitted  */
 attr = get_attr_from_code(AS_PATH_ATTR_CODE);
 peer = get_src_peer_info();
 if (!attr || !peer) return FAIL;

 my_as = peer->local_bgp_session->as;
 as_path = attr->data;
 as_path_len = attr->len;

 while (i < as_path_len) {
   i++; /* omit segment type  */
   segment_length = as_path[i++];
   for (j = 0; j < segment_length - 1; j++) {
     curr_as = get_u32(as_path + i);
     i += 4;
     if (!valley_check(next_as, curr_as)) return PLUGIN_FILTER_REJECT;
   }
 }
 next();
 return FAIL;
}
```

Retrieve data from the host implementation



Main processing of the plugin

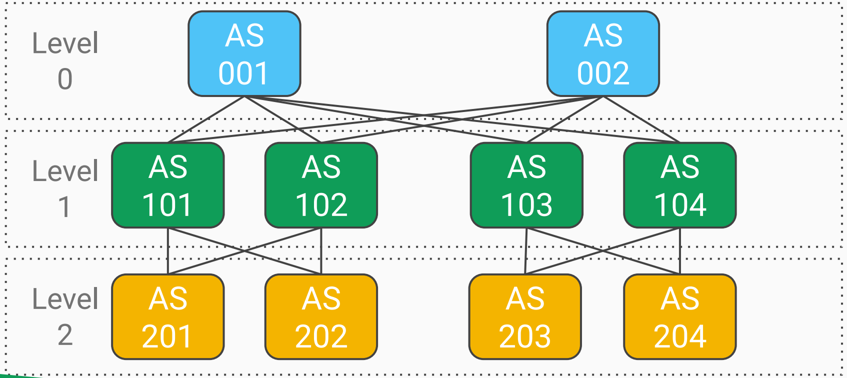
Valley Free path check with xBGP

```
uint64_t valley_free_check(args_t *args UNUSED) {
 /* variable declaration omitted  */
 attr = get_attr_from_code(AS_PATH_ATTR_CODE);
 peer = get_src_peer_info();
 if (!attr || !peer) return FAIL;

 my_as = peer->local_bgp_session->as;
 as_path = attr->data;
 as_path_len = attr->len;

 while (i < as_path_len) {
  i++; /* omit segment type  */
  segment_length = as_path[i++];
  for (j = 0; j < segment_length - 1; j++) {
   curr_as = get_u32(as_path + i);
   i += 4;
   if (!valley_check(next_as, curr_as)) return PLUGIN_FILTER_REJECT;
  }
 }
 next();
 return FAIL;
}
```

Retrieve data from the host implementation



Main processing of the plugin

The route is rejected if such a pair exists