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

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

```plaintext
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?

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

1. BGP Adj-RIB-In
   - Peer[x]
   - Import filters
   - BGP Messages from Peer

2. BGP Loc-RIB
   - All acceptable routes
   - BGP Decision Process
   - Best route(s) towards each prefix

3. BGP Adj-RIB-Out
   - Peer[y]
   - Export filters
   - BGP Messages to Peer[y]

Control plane
Data plane

FIB

RFC 4271 BGP Workflow
A plugin to support a geoloc attribute

RFC 4271 BGP Workflow
A plugin to support a geoloc attribute

My GeoLoc Plugin
- Decoding GeoLoc
- Locate nearest router
- Serializing GeoLoc

RFC 4271 BGP Workflow
A plugin to support a geoloc attribute

Decoding GeoLoc
Locate nearest router
Serializing GeoLoc

My GeoLoc Plugin

libxBGP

RFC 4271 BGP Workflow
A plugin to support a geoloc attribute

My GeoLoc Plugin
- Decoding GeoLoc
- Locate nearest router
- Serializing GeoLoc

libxBGP

RFC 4271 BGP Workflow
Agenda

- xBGP: a Paradigm Shift
- Adding a new feature with xBGP
- Uses Cases
xBGP requires a little adaptation on the host BGP implementation. We have adapted both FRRouting and BIRD to be xBGP compliant.

<table>
<thead>
<tr>
<th>Modification to the codebase</th>
<th>FRRouting (LoC)</th>
<th>BIRD Routing (LoC)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Insertion Points</td>
<td>73</td>
<td>66</td>
</tr>
<tr>
<td>Plugin API</td>
<td>624</td>
<td>415</td>
</tr>
<tr>
<td>libxbgp</td>
<td>3004 + dependencies</td>
<td></td>
</tr>
<tr>
<td>User Space eBPF VM</td>
<td>2776</td>
<td></td>
</tr>
</tbody>
</table>
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


See [https://www.pluginized-protocols.org/xbgp](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

Time xBGP - Time Native

Time Native

Lower is better

Relative Performance Impact (%)

-10

0

10

20

30

xFRRouting

xBIRD

Implementation Under Test (724k routes)

Route Reflectors
Comparison with native code

Time $xBGP - Time_{Native}$

$\times 100$

Time $Native$

Lower is better

Relative Performance Impact (%)

$\pm 30$

$\pm 20$

$\pm 10$

$\pm 0$

$\pm 10$

$\pm 20$

$\pm 30$

Route Reflectors

Origin Validation

Implementation Under Test (724k routes)
Comparison with native code

Native code uses a slower data structure

Lower is better
Valley Free path check
Valley Free path check

Level 0
- S1
- S2

Level 1
- L10
- L11
- L12
- L13

Level 2
- T20
- T21
- T22
- T23
Valley Free path check
Valley Free path check
Valley Free path check
### MyRouter Cli > show ip bgp

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

<table>
<thead>
<tr>
<th>Network</th>
<th>Next Hop</th>
<th>Metric</th>
<th>LocPref</th>
<th>Weight</th>
<th>Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>* &gt; Ec 192.168.10.0/24</td>
<td>192.168.255.20</td>
<td>0</td>
<td>100</td>
<td>0</td>
<td>100 200 i</td>
</tr>
<tr>
<td>* ec 192.168.10.0/24</td>
<td>192.168.255.4</td>
<td>0</td>
<td>100</td>
<td>0</td>
<td>100 200 i</td>
</tr>
<tr>
<td>* &gt; Ec 192.168.254.3/32</td>
<td>192.168.255.4</td>
<td>1</td>
<td>100</td>
<td>0</td>
<td>100 200 i</td>
</tr>
<tr>
<td>* ec 192.168.254.3/32</td>
<td>192.168.255.20</td>
<td>0</td>
<td>100</td>
<td>0</td>
<td>100 200 i</td>
</tr>
<tr>
<td>* &gt; Ec 192.168.254.4/32</td>
<td>192.168.255.20</td>
<td>0</td>
<td>100</td>
<td>0</td>
<td>100 200 i</td>
</tr>
</tbody>
</table>

### RFC 7938 Use of BGP for Routing in Large-Scale Data Centers

- **Level 0**
  - AS 001

- **Level 1**
  - AS 100

- **Level 2**
  - AS 200
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

<table>
<thead>
<tr>
<th>Network</th>
<th>Next Hop</th>
<th>Metric</th>
<th>LocPref</th>
<th>Weight</th>
<th>Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>* &gt;Ec 192.168.10.0/24</td>
<td>192.168.255.20</td>
<td>0</td>
<td>100</td>
<td>0</td>
<td>100 200 i</td>
</tr>
<tr>
<td>* ec 192.168.10.0/24</td>
<td>192.168.255.4</td>
<td>0</td>
<td>100</td>
<td>0</td>
<td>100 200 i</td>
</tr>
<tr>
<td>* &gt;Ec 192.168.254.3/32</td>
<td>192.168.255.4</td>
<td>1</td>
<td>100</td>
<td>0</td>
<td>100 200 i</td>
</tr>
<tr>
<td>* ec 192.168.254.3/32</td>
<td>192.168.255.20</td>
<td>0</td>
<td>100</td>
<td>0</td>
<td>100 200 i</td>
</tr>
<tr>
<td>* &gt;Ec 192.168.254.4/32</td>
<td>192.168.255.20</td>
<td>0</td>
<td>100</td>
<td>0</td>
<td>100 200 i</td>
</tr>
</tbody>
</table>

Where are these routes sourced from?

Where are these routes sourced from?
Valley Free path check with xBGP

One plugin + one topology manifest for all routers!

(81 LoC)
Valley Free path check with xBGP

```c
uint64_t valley_free_check(args_t *args UNUSED) {
    /* variable declaration omitted */
    attr = get_attr_from_code(ASPATH_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

```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
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

```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

The route is rejected if such a pair exists