xBGP: When You can’t wait for IETF and Vendors

Thomas Wirtgen, Tom Rousseaux, Quentin de Coninck, Randy Bush, Laurent Vanbever, Axel Legay, Olivier Bonaventure
Agenda

- Why bring programmability to BGP?
- Inside xBGP
- Use Cases
- Verifying xBGP extensions
- Conclusion
Routing on the Internet
Routing on the Internet

Router vendors do not have an unified interface to configure routers

```conf
bgp {
  group external-peers {
    type external;
    neighbor 172.17.1.99 {
      peer-as 2;
    }
  }
  routing-options {
    router-id 172.17.1.99;
    autonomous-system 1;
  }
}
```

eBGP

iBGP

iBGP

iBGP

router bgp 1
  router-id 172.17.1.99
  neighbor 192.168.1.2 remote-as 2
  address-family ipv4 unicast
  neighbor 192.168.1.2 activate
  exit-address-family

AS 1

AS 2

AS 3
Router vendors do not have an unified interface to configure routers

All routers do not implement the same set of functionalities

```
bgp {
   group external-peers {
      type external;
      neighbor 172.17.1.99 {
         peer-as 2;
      }
   }
   routing-options {
      router-id 172.17.1.99;
      autonomous-system 1;
   }
}
router bgp 1
   router-id 172.17.1.99
   neighbor 192.168.1.2 remote-as 2
   address-family ipv4 unicast
   neighbor 192.168.1.2 activate
   exit-address-family
```
As networks evolve, so do routing protocols

One does not simply ask to your routers vendor...

1. Standardisation of the new feature by the IETF (3.5 year in average for BGP)
2. Implementation on the vendor OSes
3. Update your routers
As networks evolve, so do routing protocols

One does not simply ask to your routers vendor...

1. Standardisation of the new feature by the IETF (3.5 year in average for BGP)
2. Implementation on the vendor OSes
3. Update your routers

You can not easily influence steps 1 and 2!

I would like to propose a new feature

We will think about how to standardize it if it adds value

Can you please update the router OS?

You don’t have the required support licence to ask us this
Current paradigm slows innovation

Problem #1: No consensus to propose a unified configuration model

Problem #2: Protocol extensions not implemented on all routers

Problem #3: Slow upgrade process

⇒ xBGP is designed to bring innovation to network engineers.
Agenda

- Why bring programmability to BGP?
- **Inside xBGP**
- Use Cases
- Verifying xBGP extensions
- Conclusion
BGP implementations are opaque
BGP implementations are opaque

Protocol

CLI

NetConf

SNMP

API

RIB
Internal Data Structure
Neighbor routers context

Protocol memory

<?xml version="1.0"?>
<rpc-reply>
  <data>
    <interfaces>
      <interface>
        <name>Gig</name>
        <type>ian
      </interface>
    </interfaces>
  </data>
</rpc-reply>
xBGP a paradigm shift

Operators can now add extension codes to their routers
xBGP propose a common interface for routers

Thanks to xBGP, the same extension code can run on several implementations

Routers are now seen as microkernels
Let’s take an example of feature. The GeoLoc TLV

BGP Path Record Attribute: draft-raszuk-idr-bgp-pr-05
How to alter BGP to make it xBGP compliant?

Let’s take an example of feature. The GeoLoc TLV

BGP UPDATE
a.b.c.0/24
via AS2

<Lat, Lon>

Add GeoLoc on the input edge routers

BGP Path Record Attribute: draft-raszuk-idr-bgp-pr-05
How to alter BGP to make it xBGP compliant?

Let’s take an example of feature. The GeoLoc TLV

1. Add GeoLoc on the input edge routers
2. Spread the GeoLoc inside the IGP

BGP Path Record Attribute: draft-raszuk-idr-bgp-pr-05
Let's take an example of feature. The GeoLoc TLV

How to alter BGP to make it xBGP compliant?

1. Add GeoLoc on the input edge routers
2. Spread the GeoLoc inside the IGP
3. Remove GeoLoc on the output edge routers

BGP Path Record Attribute: draft-raszuk-idr-bgp-pr-05
How to alter BGP to make it xBGP compliant?
How to alter BGP to make it xBGP compliant?
How to alter BGP to make it xBGP compliant?

RFC 4271 BGP Workflow
How to alter BGP to make it xBGP compliant?

RFC 4271 BGP Workflow
How to alter BGP to make it xBGP compliant?
How to alter BGP to make it xBGP compliant?

RFC 4271 BGP Workflow
How to alter BGP to make it xBGP compliant?

RFC 4271 BGP Workflow
How to alter BGP to make it xBGP compliant?

RFC 4271 BGP Workflow
How to alter BGP to make it xBGP compliant?

BGP Messages From Peers → Adj-RIB-IN → Import Filters → Loc-RIB → BGP Decision Process → Export Filters → Adj-RIB-OUT → BGP Messages To Peers

BGP Control Plane

Data Plane

FIB

RFC 4271 BGP Workflow
How to alter BGP to make it xBGP compliant?

RFC 4271 BGP Workflow

BGP Messages From Peers

Adj-RIB-IN

Import Filters

GeoLoc Ext.

Adj-RIB-OUT

Export Filters

BGP Messages To Peers

BGP Decision Process

FIB

BGP Control Plane

Data Plane
How to alter BGP to make it xBGP compliant?
How to alter BGP to make it xBGP compliant?

RFC 4271 BGP Workflow
How to alter BGP to make it xBGP compliant?

RFC 4271 BGP Workflow
How to alter BGP to make it xBGP compliant?

RFC 4271 BGP Workflow

Router Coordinates
Routing Tables
Peer states...

xBGP API

FIB

BGP Messages
From Peers

BGP Control Plane
Data Plane

BGP Messages
To Peers

Adj-RIB-IN

Import Filters

Decoding GeoLoc

Add GeoLoc

Use GeoLoc

Delete GeoLoc

Encode GeoLoc

Adj-RIB-OUT

Export Filters

XBGP API

RFC 4271 BGP Workflow
To communicate with BGP, xBGP extension codes **must** use the xBGP API.

The xBGP API contains:
- Send and Read BGP messages
- Setters & Getters (BGP routes, attributes, peer state, etc.)
- RIB access
- Utility Functions (memory, math, etc.)
Agenda

- Why bring programmability to BGP?
- Inside xBGP
  - **Use Cases**
  - Verifying xBGP extensions
- Conclusion
Demonstrating the programmability of xBGP

xBGP requires a little adaptation to 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></td>
<td></td>
</tr>
<tr>
<td>User Space eBPF VM</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

https://www.pluginized-protocols.org/xbgp
Monitoring the AS Path length

We want to count the number of ASes contained in each BGP UPDATE.

It is difficult to achieve with traditional interfaces (CLI, NetConf, Yang, etc.)

Why monitoring the AS Path?

- Filter out large AS Path
- Make analysis
uint64_t count_as_path(args_t *args) {

Monitoring the AS Path length

```c
uint64_t count_as_path(args_t *args) {
    unsigned int as_number = 0, segment_length;
    unsigned int *attribute_code = get_arg(ARG_CODE);
    unsigned int *as_path_len = get_arg(ARG_LENGTH);
    unsigned char *as_path = get_arg(ARG_DATA);

    if (!as_path || !as_path_len || !attribute_code) {
        // unable to fetch data from host implementation
        return EXIT_FAILURE;
    } else if (*attribute_code != AS_PATH_ATTR_ID) {
        return EXIT_FAILURE;
    }
    // Retrieve data from the host implementation
}
```
uint64_t count_as_path(args_t *args) {
    unsigned int as_number = 0, segment_length;
    unsigned int *attribute_code = get_arg(ARG_CODE);
    unsigned int *as_path_len = get_arg(ARG_LENGTH);
    unsigned char *as_path = get_arg(ARG_DATA);

    if (!as_path || !as_path_len || !attribute_code) {
        // unable to fetch data from host implementation
        return EXIT_FAILURE;
    } else if (*attribute_code != AS_PATH_ATTR_ID) {
        return EXIT_FAILURE;
    }

    // core part of the plugin
    while (i < *as_path_len) {
        segment_length = as_path[i + 1];
        as_number += segment_length;
        i += (segment_length * 4) + 2;
    }

    return as_number;
}

Retrieve data from the host implementation
Parse the AS-PATH attribute
uint64_t count_as_path(args_t *args) {
    unsigned int as_number = 0, segment_length;
    unsigned int *attribute_code = get_arg(ARG_CODE);
    unsigned int *as_path_len = get_arg(ARG_LENGTH);
    unsigned char *as_path = get_arg(ARG_DATA);

    if (!as_path || !as_path_len || !attribute_code) {
        // unable to fetch data from host implementation
        return EXIT_FAILURE;
    } else if (*attribute_code != AS_PATH_ATTR_ID) {
        return EXIT_FAILURE;
    }

    // core part of the plugin
    while (i < *as_path_len) {
        segment_length = as_path[i + 1];
        as_number += segment_length;
        i += (segment_length * 4) + 2;
    }

    // log the message. If it fails, returns an error code
    if (log_msg(L_INFO "as_count:%d\n", LOG_UINT(as_number)) != 0) {
        return EXIT_FAILURE;
    }
    return EXIT_SUCCESS;
}
Monitoring the AS Path length

RFC 4271 BGP Workflow
Valley Free path check

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

Diagram showing the AS levels and path selection.
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?

RFC7938 Use of BGP for Routing in Large-Scale Data Centers
Valley Free path check with xBGP

One plugin + one topology manifest for all routers!

(81 LoC)
The Path Selection Service

“Source AS” cannot influence “Transit 2” route selection
“Source AS” can not influence “Transit 2” route selection
“Source AS” can not influence “Transit 2” route selection
The Path Selection Service

“Source AS” can not influence “Transit 2” route selection

“Choose the routes according to the ranking of Source network”
Detecting BGP Zombies
Detecting BGP Zombies

UPDATE
I can reach p
Detecting BGP Zombies

UPDATE
I can reach p

UPDATE
I can reach p

p
Detecting BGP Zombies

UPDATE
I can reach p

UPDATE
I can reach p

p

p
Detecting BGP Zombies

UPDATE
I can reach p

UPDATE
I can reach p

WITHDRAW

UPDATE

WITHDRAW
Detecting BGP Zombies

UPDATE
I can reach p

WITHDRAW
p

UPDATE
I can reach p
Detecting BGP Zombies

UPDATE
I can reach p

WITHDRAW
p is still in the FIB!
Detecting BGP Zombies

UPDATE
I can reach p

UPDATE
I can reach p

WITHDRAW

p is still in the FIB!

Checks routes older than
<x> <unit of time>
Detecting BGP Zombies

UPDATE
I can reach p

UPDATE
I can reach p

WITHDRAW
p

p is still in the FIB!

Checks routes older than
<x> <unit of time>

Ask the upstream router to confirm if the route is still valid

p since 4w 7h 36m 2s
Detecting BGP Zombies

RFC 4271 BGP Workflow
Agenda

- Why bring programmability to BGP?
- Inside xBGP
- Use Cases
- **Verifying xBGP extensions**
- Conclusion
Executing arbitrary code is dangerous

The code executed by xBGP is untrusted. Could it break BGP?

The code should satisfy:
1. Termination
2. Memory Isolation
3. BGP Syntax
4. API Restriction
How to verify the properties?

The code should be annotated manually, and then passed to the verification tools. The code logic may also be modified to pass the verification checks.

Diagram:

- Extension source code
- SeaHorn
- CBMC
- llvm/clang
- Extension bytecode
- Offline verification tools

T2
The right tool to the right property

**T2**
- Termination property

**CBMC**
- Buffer overflow
- Memory isolation
- Memory leak
- Conversion errors
- ...

**libxBGP**
- xBGP API restriction
  (offline)

**SeaHorn**
- BGP Related properties
  (i.e. BGP syntax)

Extension codes are guaranteed to not violate the properties we defined
Example: verifying the BGP syntax

If the xBGP extension adds Geographic coordinates, it must respect the TLV format defined in the draft.

attribute.type.flags.optional == 1
attribute.type.flags.transitive == 0
attribute.type.flags.partial == 0
attribute.type.flags.extended == 0

attribute.type.code == GeoTLV Identifier

attribute.length == 8

lo_latitude <= attribute.data.latitude <= hi_latitude
lo_longitude <= attribute.data.longitude <= hi_longitude
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

xBGP provides new opportunities with other routing protocols

thomas.wirtgen@uclouvain.be
Backup Slides
Using several tools is cumbersome

We propose a kind of DSL that unifies the annotations of all verification tools

Extension code

Right flags to trigger the right DSL annotations

Verification tool compiler

Verification tool solver

SAT

UNSAT