# yabgp Documentation

*Release 0.8.1*

**Peng Xiao**

**Sep 12, 2019**

# Contents

YABGP is a yet another Python implementation for BGP Protocol. It can be used to establish BGP connections with all kinds of routers (include real Cisco/HuaWei/Juniper routers and some router simulators like GNS3) and receive/parse BGP messages for future analysis.

Support sending BGP messages(route refresh/update) to the peer through RESTful API. YABGP can't send any BGP update messages by itself, it's just a agent, so there can be many agents and they can be controlled by a contoller.

# CHAPTER 1

## Table of Contents

## 1.1 Features

- It can establish BGP session based on IPv4 address (TCP Layer) in active mode(as TCP client);
- Support TCP MD5 authentication(IPv4 and does not support Windows now);
- BGP capabilities support: 4 Bytes ASN, Route Refresh(Cisco Route Refresh), Add Path send/receive;
- Address family support:
  - IPv4/IPv6 unicast
  - IPv4 Flowspec(limited)
  - IPv4 sr-policy(limited)
  - IPv4/IPv6 MPLSVPN
  - EVPN (partially supported)
- Decode all BGP messages to json format and write them into files in local disk(configurable);
- Support basic RESTFUL API for getting running information and sending BGP messages.
- Platform support: Linux/Unix(recommended), Mac OS and Windows.

**Note:** yabgp is a light weight BGP agent used for connecting network devices. It only can be TCP client in one BGP peering connection and can't send any update messages by itself(send through REST API). We recommend that each yabgp process connect only one BGP neighbor, so each process is independent with each other, we can start many yabgp processes within the same machine or in different machines. There can be a central controller which can controll all yabgp processes through REST API to send BGP update messages.

## 1.2 Installation

We recommend run `yabgp` through python virtual-env from source code or pip install

### 1.2.1 From source code

Use yabgp from source code:

```
$ virtualenv yabgp-virl
$ source yabgp-virl/bin/activate
$ git clone https://github.com/smartbgp/yabgp
$ cd yabgp
$ pip install -r requirements.txt
$ cd bin
$ python yabgpd -h
```

### 1.2.2 From pip

Use pip install

```
$ virtualenv yabgp-virl
$ source yabgp-virl/bin/activate
$ pip install yabgp
$ which yabgpd
/home/yabgp/yabgp-virl/bin/yabgpd
$ yabgpd -h
```

**Note:** For `virtualenv`, you can install it from pip. And make sure you have installed `python-dev` based on your operation system, for example Ubuntu, you can install it from `apt-get install python-dev`. otherwise, you may get error when install requirement from requirements.txt

## 1.3 Tutorial

### 1.3.1 Basic Usage

We can use `yabgpd` help.

```
$ yabgpd -h
```

The simple way to start a yabgp agent is (There are four mandatory parameters):

```
$ yabgpd --bgp-local_addr=10.75.44.11 --bgp-local_as=23650 \
         --bgp-remote_addr=10.124.1.245 --bgp-remote_as=23650
```

The default address family will be IPv4 unicast. If you need support other address family, you can use something like:

` $ yabgpd --bgp-local_addr=10.75.44.122 --bgp-local_as=100 --bgp-remote_addr=10.75.195.199 --bgp-remote_as=100 --bgp-afi_safi=ipv4_srte,flowspec,ipv4,bgpls `

that session will suport: IPv4 Unicast, IPv4 Flowspec, IPv4 SR TE and BGP linkstate.

### 1.3.2 Configuration

The configuration sample can be found at https://github.com/smartbgp/yabgp/blob/master/etc/yabgp/yabgp.ini.sample

### 1.3.3 Advanced Usage

If you change the default setting (like change the `write_dir`), please start the `yabgp` use the configuration file:

```
$ cp etc/yabgp/yabgp.ini.sample etc/yabgp/yabgp.ini
$ yabgpd --bgp-local_addr=10.75.44.11 --bgp-local_as=23650 --bgp-remote_addr=10.124.1.
↪245 \
         --bgp-remote_as=23650 --bgp-md5=cisco --config-file=../etc/yabgp/yabgp.ini
```

### 1.3.4 Logging and Debug

The default setting is loggint to console, if you want to write log files and no console output, please use:

```
$ yabgpd --bgp-local_addr=10.75.44.11 --bgp-local_as=23650 --bgp-remote_addr=10.124.1.
↪245 \
         --bgp-remote_as=23650 --bgp-md5=cisco --nouse-stderr --log-file=test.log
```

If you want to change the log level to debug, use *–verbose* option.

```
$ yabgpd --bgp-local_addr=10.75.44.11 --bgp-local_as=23650 --bgp-remote_addr=10.124.1.
↪245 \
         --bgp-remote_as=23650 --bgp-md5=cisco --verbose
```

## 1.4 BGP Message Format

### 1.4.1 Basic

BGP message is **json** format, and it has four keys: `t`, `seq`, `type` and `msg`.

`t` is timestamp, it is a standard unix time stamp, the number of seconds since 00:00:00 UTC on January 1, 1970.

Example:

```
timestamp = 1372051151.2572551 # that is Mon Jun 24 13:19:11 2013.
```

`seq` is sequence number It is a interger and represents the message number.

`type` It is a interger that represents the message type.

there are eight types of message now.

| Type | Description |
|---|---|
| 0 | Session information Message |
| 1 | BGP Open Message |
| 2 | BGP Update Message |
| 3 | BGP Notification Message |
| 4 | BGP Keepalive Message |
| 5 | BGP Route Refresh Message |
| 128 | BGP Cisco Route Refresh Message |
| 6 | Malformed Update Message |

`msg` is the message contents, and its format is different according to different message types.

### 1.4.2 Session Info Message

Session information message (type = 0) always represents the TCP connection is closed by some reason, for example, the connection is refused or the connection is reset by the other site.

```
{
    "t": 1372065358.666234,
    "seq": 12,
    "type": 0,
    "msg": "Connection lost:Connection to the other side was lost in a non-clean␣
↪fashion."
}
{
    "t": 1451360038.041204,
    "seq": 1,
    "type": 0,
    "msg": "Client connection failed: Couldn't bind: 49: Can't assign requested␣
↪address."
}
```

### 1.4.3 Open Message

BGP Open message (type = 1). for the meaning of the keys, please check RFC 4271 section 4.2.

```
{
    "t": 1452008814.016207,
    "seq": 1,
    "type": 1,
    "msg":{
        "hold_time": 180,
        "capabilities": {
            "cisco_route_refresh": true,
            "route_refresh": true,
            "graceful_restart": true,
            "add_path": "ipv4_both",
            "four_bytes_as": true,
            "afi_safi": [[1, 1], [1, 133]]
        },
        "bgp_id": "9.9.9.9",
        "version": 4,
        "asn": 9308
```

(continues on next page)

(continued from previous page)

```
    }
}
```

### 1.4.4 Update Message

BGP Update message (type = 2), the value of `msg` for a update message is a dict, and it has four keys : `attr`, `nlri`, `withdraw` and `afi_safi`.

| Key | Value | Description |
|---|---|---|
| "attr" | dict | Path Attributes |
| "nlri" | list | Network Layer Reachability Information |
| "withdraw" | list | Withdrawn Routes |
| "afi_safi" | string | address family |

simple format:

```
{
    "msg":{
        "attr": {},
        "withdraw": [],
        "nlri": [],
        "afi_safi": "ipv4"
    }
}
```

and here is a full BGP update message example:

```
{
    "t": 1450668281.624188,
    "seq": 17,
    "type": 2,
    "msg": {
        "attr": {
            "1": 0,
            "2": [[2, [209, 2768, 2768, 2768, 2768]]],
            "3": "1.1.1.2",
            "5": 500,
            "8": ["1234:5678", "2345:6789"],
            "9": "1.1.1.2",
            "10": ["1.1.1.1", "2.2.2.2", "3.3.3.3"]
        },
        "nlri": ["65.122.75.0/24", "65.122.74.0/24"],
        "withdraw": [],
        "afi_safi": "ipv4"
    }
}
```

and here is a withdraw message:

```
{
    "t": 1450163221.123568,
    "seq": 17,
    "type": 2,
```

(continues on next page)

(continued from previous page)

```
    "msg": {
        "attr": {},
        "nlri": [],
        "withdraw": ["65.122.75.0/24", "65.122.74.0/24"]
        "afi_safi": "ipv4"
    }
}
```

The `withdraw` and `nlri` are all List, they contain the particular prefix string. Here is one real BGP decoded message example

Example for a `nlri` or `withdraw` value:

```
["1.1.1.1/32", "2.2.2.2/32"]
```

The value of key `attr` is a dictionary. it contains the BGP prefix's attribute, the dict's key represent what of kind of attribute, and the value is this attribute's value.

The attribute we supported now is: (reference by IANA)

```
{
    "1": "ORIGIN",
    "2": "AS_PATH",
    "3": "NEXT_HOP",
    "4": "MULTI_EXIT_DISC",
    "5": "LOCAL_PREF",
    "6": "ATOMIC_AGGREGATE",
    "7": "AGGREGATOR",
    "8": "COMMUNITY",
    "9": "ORIGINATOR_ID",
    "10": "CLUSTER_LIST",
    "14": "MP_REACH_NLRI",
    "15": "MP_UNREACH_NLRI",
    "16": "EXTENDED_COMMUNITY",
    "17": "AS4_PATH",
    "18": "AS4_AGGREGATOR",
    "22": "PMSI_TUNNEL",
    "23": "TUNNEL_ENCAPSULATIONS",
    "128": "ATTR_SET"
}
```

Example for `attr` value:

```
{
    "1": 0,
    "2": [[2, [209, 2768, 2768, 2768, 2768]]],
    "3": "1.1.1.2",
    "5": 500,
    "8": ["1234:5678", "5678:1234"],
    "9": "1.1.1.2",
    "10": ["1.1.1.1", "2.2.2.2", "3.3.3.3"]
}
```

Next, we will explain the detail structure of each attribute.

### 1. ORIGIN

`ORIGIN` value is an interger, has three kinds of value (0, 1, 2 ). it defines the origin of the path information. The data octet can assume the following values:

| Value | Meaning |
|---|---|
| 0 | IGP |
| 1 | EGP |
| 2 | INCOMPLETE |

### 2. AS_PATH

`AS_PATH` value is a list, it has one item at least, each item also is a list and it reprensents one `AS PATH` segment,like [[sgement_1], [segment_2], ……], and each AS path segment is represented by [path segment type, path segment value]. For path sgement value, its a list of interger.

each segment's first item is segment type, it has four kinds of vlaue.

| Value | Meaning |
|---|---|
| 1 | AS_SET: unordered set of ASes a route in the UPDATE message has traversed |
| 2 | AS_SEQUENCE: ordered set of ASes a route in the UPDATE message has traversed |

For example:

```
{
    "attr": {
        "2": [[2, [209, 2768, 2768, 2768, 2768]]]
    }
}
```

For this example, it only has one AS path segment: `[2, [209, 2768, 2768, 2768, 2768]]`, this segment's type is `AS_SEQUENCE`, and its value is `[209, 2768, 2768, 2768, 2768]`.

### 3. NEXT_HOP

`NEXT_HOP` is one a string, IPv4 address format, eg: '10.0.0.1'.

### 4. MULTI_EXIT_DISC

`MULTI_EXIT_DISC` is an interger.

### 5. LOCAL_PREF

`LOCAL_PREF` is an interger.

### 6. ATOMIC_AGGREGATE

`ATOMIC_AGGREGATE` is one empty string, `""`.

### 7. AGGREGATOR

AGGREGATOR is a list, it has two items, [asn, aggregator], the first is AS number, the second is IP address. eg:

```
{
    "attr": {
        "7": [100, "1.1.1.1"]
    }
}
```

### 8. COMMUNITY

COMMUNITY is a list, each item of this List is a string.

eg:

```
{
    "attr": {
        "8": ["NO_EXPORT", "1234:5678"]
    }
}
```

There are two kinds of COMMUNITY, first is "Well-Konwn", second is "The Others".

"Well-known" COMMUNITY

```
planned_shut                = 0xFFFF0000
accept_own                  = 0xFFFF0001
ROUTE_FILTER_TRANSLATED_v4  = 0xFFFF0002
ROUTE_FILTER_v4             = 0xFFFF0003
ROUTE_FILTER_TRANSLATED_v6  = 0xFFFF0004
ROUTE_FILTER_v6             = 0xFFFF0005
NO_EXPORT                   = 0xFFFFFF01
NO_ADVERTISE                = 0xFFFFFF02
NO_EXPORT_SUBCONFED         = 0xFFFFFF03
NOPEER                      = 0xFFFFFF04
```

### 9. ORIGINATOR_ID

ORIGINATOR_ID is a string, format as IPv4 address, just NEXT_HOP eg: "10.0.0.1".

### 10. CLUSTER_LIST

CLUSTER_LIST is a list, each item in this List is a string, format as IPv4 address. eg:

```
{
    "attr": {
        "10": ["1.1.1.1", "2.2.2.2", "3.3.3.3"]
    }
}
```

### 11. MP_REACH_NLRI

**Note:** Only No IPv4 Unicast BGP Update messages have the attributes `MP_REACH_NLRI` and `MP_UNREACH_NLRI`, because for IPv4 Unicast, its NLRI and WITHDRAW informations are contain in `nlri` and `withdraw` value. So for No IPv4 Unicast BGP messages, its `nlri` and `withdraw` are empty, and its own nlri and withdraw information contains in `MP_REACH_NLRI` and `MP_UNREACH_NLRI`.

`MP_REACH_NLRI` is one complex dict which has three key `afi_safi`, `next_hop`, `nlri`. and according to differences between the `afi_safi`, the Data structure of `next_hop` and `nlri` are different.

`afi_safi` value and meanings, reference by Address Family Numbers and Subsequent Address Family Identifiers (SAFI) Parameters

In addition to IPv4 Unicast, here are the `afi_safi` we support:

| Value | Meaning |
|---|---|
| [1, 128] | IPv4 MPLSVPN |
| [1, 133] | IPv4 Flowspec |
| [1, 73] | IPv4 Sr-policy |
| [2, 1] | IPv6 Unicast |
| [2, 128] | IPv6 MPLSVPN |
| [25, 70] | L2VPN EVPN |
| ... | ... |

#### IPv4 MPLSVPN

```
{
    "attr":{
        "14": {
            "afi_safi": [1, 128],
            "nexthop": {"rd": "0:0", "str": "2.2.2.2"},
            "nlri": [
                {
                    "label": [25],
                    "rd": "100:100",
                    "prefix": "11.11.11.11/32"}]}
        }
}
```

#### IPv4 FlowSpec

```
{
    "attr":{
        "14": {
            "afi_safi": [1, 133],
            "nexthop": "",
            "nlri": [
                {"1": "192.88.2.3/24", "2": "192.89.1.3/24", "5": "=80|=8080"},
                {"1": "192.88.5.3/24", "2": "192.89.2.3/24", "5": "=80|=8080"}
            ]
        }
```

(continues on next page)

(continued from previous page)

```
    }
}
```

The nlri contains filters and values, and the supported filters are:

```
BGPNLRI_FSPEC_DST_PFIX = 1   # RFC 5575
BGPNLRI_FSPEC_SRC_PFIX = 2   # RFC 5575
BGPNLRI_FSPEC_IP_PROTO = 3   # RFC 5575
BGPNLRI_FSPEC_PORT = 4   # RFC 5575
BGPNLRI_FSPEC_DST_PORT = 5   # RFC 5575
BGPNLRI_FSPEC_SRC_PORT = 6   # RFC 5575
BGPNLRI_FSPEC_ICMP_TP = 7   # RFC 5575
BGPNLRI_FSPEC_ICMP_CD = 8   # RFC 5575
BGPNLRI_FSPEC_TCP_FLAGS = 9   # RFC 5575
BGPNLRI_FSPEC_PCK_LEN = 10   # RFC 5575
BGPNLRI_FSPEC_DSCP = 11   # RFC 5575
BGPNLRI_FSPEC_FRAGMENT = 12   # RFC 5575
```

The value format of each filter are: *BGPNLRI_FSPEC_DST_PFIX* and *BGPNLRI_FSPEC_SRC_PFIX* are prefixes format, others are integers, but in string format like:

*=80* means equal to *80*

*=80|=8080* means equal to 80 or 8080.

*>=80|<40* means geater than 80 or equal to 80 or less than 40

### IPv4 Sr-policy

```
{
    "attr": {
        "8": ["NO_ADVERTISE"],
        "14": {
            "afi_safi": [1, 73],
            "nexthop": "192.168.5.5",
            "nlri": {
                "distinguisher": 0,
                "color": 10,
                "endpoint": "192.168.76.1"
            }
        },
        "16": ["route-target:10.75.195.199:00"],
        "23": {
            "0": "new",
            "12": 100,
            "13": 25102,
            "128": [
                {
                    "9": 10,
                    "1": [
                        {
                            "1": {
                                "label": 2000,
                                "TC": 0,
                                "S": 0,
                                "TTL": 255
```

(continues on next page)

(continued from previous page)

```
                            }
                        },
                        {
                            "3": {
                                "node": "10.1.1.1",
                                "SID": {
                                    "label": 3000,
                                    "TC": 0,
                                    "S": 0,
                                    "TTL": 255
                                }
                            }
                        }
                    ]
                }
            ]
        }
    }
}
```

attribute explaination(only for this format):

```
"8": Optionally assign
"14": Multiprotocol Reacable Attribute
"16": Route Target Extended Community
"23": Tunnel Encapsulation Attribute
    "0": if the ios version lower than 6.4.1.14(Cisco facility), the value should be
↪'old', and in the meantime,
         the key of Preference should be '6', key of Binding SID should be '7', else␣
↪it should be 'new' and key
         of Preference and Binding SID should be '12' and '13'
    "6"/"12": Preference
    "7"/"13": Binding SID
    "128": Multiple segement lists
        "9": Weighted
        "1": Segement list
            "1": Segement type 1
                "label": Value of MPLS Label
                "TC": Assign optionally, default value is 0
                "S": Assign optionally, default value is 0
                "TTL": Assign optionally, default value is 255
            "3": Segement type 3
                "node": An Ipv4 Address
                "SID": Assign Optionally, inner structure similar to Segement type 1
```

### IPv6 Unicast

For IPv6 Unicast, it has three or four keys:

```
{
    "attr":{
        "14": {
            "afi_safi": [2, 1],
            "linklocal_nexthop": "fe80::c002:bff:fe7e:0",
            "nexthop": "2001:db8::2",
```

(continues on next page)

(continued from previous page)

```
            "nlri": ["::2001:db8:2:2/64", "::2001:db8:2:1/64", "::2001:db8:2:0/64"]}
    }
}
```

The value of the Length of Next Hop Network Address field on a `MP_REACH_NLRI` attribute shall be set to 16, when only a global address is present, or 32 if a link-local address is also included in the Next Hop field.

### IPv6 MPLSVPN

```
{
    "attr":{
        "14": {
            "afi_safi": [2, 128],
            "nexthop": {"rd": "100:12", "str": "::ffff:172.16.4.12"},
            "nlri": [
                {
                    "label": [54],
                    "rd": "100:12",
                    "prefix": "2010:0:12:4::/64"},
                {
                    "label": [55],
                    "rd": "100:12",
                    "prefix": "2010:1:12::/64"
                }
            ]
        },
        "16": ["route-target:100:12"]
        }
}
```

### EVPN

```
{
    "attr": {
        "1": 0,
        "2": [],
        "5": 100,
        "14": {
            "afi_safi": [25, 70],
            "nexthop": "10.75.44.254",
            "nlri": [
                {
                    "type": 2,
                    "value": {
                        "eth_tag_id": 108,
                        "ip": "11.11.11.1",
                        "label": [0],
                        "rd": "172.17.0.3:2",
                        "mac": "00-11-22-33-44-55",
                        "esi": 0}}]
        }
    }
}
```

### 12. MP_UNREACH_NLRI

The difference between `MP_REACH_NLRI` and `MP_UNREACH_NLRI` is that `MP_UNREACH_NLRI` only has two keys, `afi_safi` and `withdraw`, and there structure is the same.

#### IPv4 MPLSVPN

```
{
    "attr":{
        "15": {
            "afi_safi": [1, 128],
            "withdraw": [
                {
                    "rd": "100:100",
                    "prefix": "11.11.11.11/32"}]}
        }
}
```

#### IPv4 FlowSpec

```
{
    "attr":{
        "15": {
            "afi_safi": [1, 133],
            "withdraw": [
                {"1": "192.88.2.3/24", "2": "192.89.1.3/24", "5": "=80|=8080"},
                {"1": "192.16.0.0/8", "6": "=8080"}
            ]
        }
    }
}
```

#### IPv4 Sr-policy

```
{
    "attr":{
        "15": {
            "afi_safi": [1, 73],
            "withdraw": {
                "distinguisher": 0,
                "color": 10,
                "endpoint": "192.168.76.1"
            }
        }
    }
}
```

### IPv6 Unicast

```
{
    "attr":
        "15": {
            "afi_safi": [2, 1],
            "withdraw": ["::2001:db8:2:2/64", "::2001:db8:2:1/64", "::2001:db8:2:0/64
↪"]}
}
```

### IPv6 MPLSVPN

```
{
    "attr":{
        "15": {
            "afi_safi": [2, 128],
            "withdraw": [
                {
                    "label": [54],
                    "rd": "100:12",
                    "prefix": "2010:0:12:4::/64"},
                {
                    "label": [55],
                    "rd": "100:12",
                    "prefix": "2010:1:12::/64"
                }
            ]}
}}
```

### EVPN

```
{
    "attr": {
        "15": {
            "afi_safi": [25, 70],
            "withdraw": [
                {
                    "type": 2,
                    "value": {
                        "eth_tag_id": 108,
                        "ip": "11.11.11.1",
                        "label": [0],
                        "rd": "172.17.0.3:2",
                        "mac": "00-11-22-33-44-55",
                        "esi": 0}}]
        }
    }
}
```

### 13. EXTENDED_COMMUNITY

Extended community we supported:

```
#  VPN Route Target  #
route-target  # Route Target

# Route Origin (SOO site of Origin)
route-origin  # Route Origin

# BGP Flow Spec
redirect-nexthop  # redirect to ipv4/v6 nexthop
traffic-rate  # traffic-rate
redirect-vrf  # redirect Route Target
traffic-marking-dscp  # traffic-marking DSCP value

# Transitive Opaque
color # Color, treated like color-00, leftmost 2 bits of reserved field = 00, CO bits␣
↪= 00
# Color, leftmost 2 bits of reserved field = 00, CO bits = 00
# srpolicy -> IGP
color-00
# Color, leftmost 2 bits of reserved field = 01, CO bits = 01
# srpolicy -> same afi null endpoint -> any null endpoint -> IGP
color-01
# Color, leftmost 2 bits of reserved field = 10, CO bits = 10
# srpolicy -> same afi null endpoint -> any null endpoint -> same afi endpoint -> any␣
↪endpoint -> IGP
color-10
# Color, leftmost 2 bits of reserved field = 11, CO bits = 11
# treated like color-00
color-11
encapsulation  # encapsulation

# BGP EVPN
mac-mobility  # Mac Mobility
esi-label  # ESI MPLS Label
es-import  # ES Import
router-mac  # EVPN Router MAC
```

### 14. AS4_PATH

4 bytes AS PATH same as `AS_PATH`.

### 15. AS4_AGGREGATOR

4 bytes AS same as `AGGREGATOR`.

### 16. PMSI_TUNNEL

"P-Multicast Service Interface Tunnel (PMSI Tunnel) attribute". This is an optional transitive BGP attribute. The format of this attribute is defined as follows:

```
{
    "mpsl_label": 625,
    "tunnel_id": "4.4.4.4",
    "tunnel_type": 6,
```

(continues on next page)

(continued from previous page)

```
    "leaf_info_required": 0
}
```

### 1.4.5 Notification Message

BGP notification message is type 3.

```
{
    "t": 1452236692.201259,
    "seq": 28,
    "type": 3,
    "msg": {
        "data": "'\\x03\\xe8'",
        "sub_error": "Bad Peer AS",
        "error": "OPEN Message Error"
    }
}
```

### 1.4.6 Keepalive Message

BGP Keepalive message type is 4. Example:

```
{
    "t": 1372065358.666234,
    "seq": 11,
    "type": 4,
    "msg": null
}
```

### 1.4.7 Route Refresh Message

Route refresh message content is (AFI, SAFI).

```
{
    "t": 1452237198.880322,
    "seq": 10,
    "type": 5,
    "msg": {
        "res": 0,
        "afi": 1,
        "safi": 1
    }
}
```

### 1.4.8 Cisco Route Refresh Message

```
{
    "t": 1452237198.880322,
    "seq": 10,
```

(continues on next page)

(continued from previous page)

```
    "type": 128,
    "msg": {
        "res": 0,
        "afi": 1,
        "safi": 1
    }
}
```

### 1.4.9 Malformed Update Message

If the BGP update message's encoding is wrong and some part of it can't be decoded, then it will write this message as malformed update message, for example:

```
{
    "t": 1452237406.457384,
    "seq": 21,
    "type": 6,
    "msg":{
        "attr": null,
        "nlri": ["200.0.0.0/24", "201.0.0.0/24"],
        "withdraw": [],
        "hex": "hex": "'\\x00\\x00\\x00*@\\x01\\x01\\x00@\\x02\\x0e\\x02\\x03\\x00\\"
    }
}
```

## 1.5 RESTFUL API

yabgp provides various endpoints that can be used to interact with the data and routers. Web API uses JSON format.

REST API has basic http auth, through usename and password. The usename and password configured in configuration file and they have default values.

```
[rest]
# Address to bind the API server to.
# bind_host = 0.0.0.0

# Port the bind the API server to.
# bind_port = 8801

# username and password for api server
# username = admin
# password = admin
```

### 1.5.1 Version 1

The REST API lives at the `/v1` endpoint and responsds to GET, POST, PUT, and DELETE.

#### Root

Used to check the API status, if it can work.

**Example request**:

```
GET /v1
```

**Example response**:

It will return the status of this version of API.

```
{
  "status": "stable",
  "updated": "2015-01-22T00:00:00Z",
  "version": "v1"
}
```

### Peer's Information

Try to get peer's configuration and running information.

### One Peers

Get one peer's configuration and running information.

```
GET /v1/peer/<peer_ip_address>/state
```

**Example response**:

```
{
    "peer": {
        "fsm": "ESTABLISHED",
        "local_addr": "10.75.44.11",
        "local_as": 23650,
        "remote_addr": "10.124.1.245",
        "remote_as": 23650,
        "uptime": 7.913731813430786
    }
}
```

### Get send/recieve version

Get peer's send/recieve version .

```
GET /v1/peer/<peer_ip_address>/version/<action>
action : send, received
```

**Example response**:

```
{
    "version": {
        "flowspec": 0,
        "ipv4": 0,
        "mpls_vpn": 0,
        "sr_policy": 0
    }
}
```

### Message Statistic

Get the BGP message sending and receiving number statistic.

```
GET /v1/peer/<peer_ip_address>/statistic
```

**Example response**:

```
{
    "receive": {
        "Keepalives": 3,
        "Notifications": 0,
        "Opens": 1,
        "Route Refresh": 0,
        "Updates": 5
    },
    "send": {
        "Keepalives": 3,
        "Notifications": 0,
        "Opens": 1,
        "Route Refresh": 0,
        "Updates": 0
    }
}
```

### Send Message

### Route Refresh

Send BGP route refresh message to peer.

| afi | safi | Description |
|---|---|---|
| 1 | 1 | IPv4 unicast |
| 1 | 128 | IPv4 MPLS VPN |
| 2 | 1 | IPv6 unicast |
| 2 | 128 | IPv6 MPLS VPN |

```
POST /v1/peer/<peer_ip_address>/send/route-refresh
```

POST data format

```
{
    "afi": 1,
    "safi": 1,
    "res": 0
}
```

**Example response**:

```
{
    "status": true
}
```

### Update

Send BGP update message to peer

```
POST /v1/peer/<peer_ip_address>/send/update
```

POST data format for update

```
{
    "attr":{
        "1": 0,
        "2": [],
        "3": "192.0.2.1",
        "5": 100,
        "8": ["NO_EXPORT"]
},
    "nlri": ["172.20.1.0/24", "172.20.2.0/24"]
}
```

POST data format for withdraw

```
{
    "withdraw": ["172.20.1.0/24", "172.20.2.0/24"]
}
```

**Example response**:

```
{
    "status": true
}
```

### Manual start and stop

#### manual start

Try to manual start BGP session

```
GET /v1/peer/<peer_ip_address>/manual-start
```

**Example response**:

```
{
    "status": true
}
```

#### manual stop

Try to manual start BGP session

```
GET /v1/peer/<peer_ip_address>/manual-stop
```

**Example response**:

```
{
    "status": true
}
```

### Adj Rib In Search

search Adj Rib In (received RIB) based on address family.

### IPv4 unicast

```
POST /v1/peer/{{peer}}/adj-rib-in?afi_safi=ipv4
```

POST data format:

```
{
    "data": ["186.96.174.0","199.36.240.0/22"]
}
```

`data` is prefix or IP list, for prefix format, we use exactly match, for IP format, we use prefix longest match

The response example: (each ip/prefix match results with attributes and prefix)

```
{
    "data": {
        "186.96.174.0": {
            "attr": {
                "1": 0,
                "2": [
                    [
                        2,
                        [
                            27418,
                            8100,
                            3491,
                            3356,
                            3549,
                            7049,
                            11750
                        ]
                    ]
                ],
                "3": "10.75.44.254",
                "5": 100,
                "8": [
                    "3356:3",
                    "3356:86",
                    "3356:575",
                    "3356:666",
                    "3356:2003",
                    "3549:4813",
                    "3549:34032",
                    "8100:50",
                    "29761:50"
                ],
```

(continues on next page)

(continued from previous page)


```
                "9": "172.17.0.7",
                "10": [
                    "2.2.2.2"
                ]
            },
            "prefix": "186.96.174.0/24"
        },
        "199.36.240.0/22": {
            "attr": {
                "1": 0,
                "2": [
                    [
                        2,
                        [
                            27418,
                            8100,
                            174,
                            13238
                        ]
                    ]
                ],
                "3": "10.75.44.254",
                "5": 100,
                "7": [
                    13238,
                    "100.43.92.159"
                ],
                "8": [
                    "174:21001",
                    "174:22013",
                    "8100:50",
                    "29761:50"
                ],
                "9": "172.17.0.7",
                "10": [
                    "2.2.2.2"
                ]
            },
            "prefix": "199.36.240.0/22"
        }
    },
    "status": true
}
```


### Adj Rib out Search

search Adj Rib out (sended RIB) based on address family.

### IPv4 unicast

```
POST /v1/peer/{{peer}}/adj-rib-out?afi_safi=ipv4
```

POST data format:

```
{
    "data": ["186.96.174.0/24","199.36.240.0/22"]
}
```

`data` is prefix list.

The response example: (each ip/prefix match results with attributes and prefix)

```
{
    "data": {
        "186.96.174.0/24": {
            "attr": {
                "1": 0,
                "2": [
                    [
                        2,
                        [
                            27418,
                            8100,
                            3491,
                            3356,
                            3549,
                            7049,
                            11750
                        ]
                    ]
                ],
                "3": "10.75.44.254",
                "5": 100,
                "8": [
                    "3356:3",
                    "3356:86",
                    "3356:575",
                    "3356:666",
                    "3356:2003",
                    "3549:4813",
                    "3549:34032",
                    "8100:50",
                    "29761:50"
                ],
                "9": "172.17.0.7",
                "10": [
                    "2.2.2.2"
                ]
            },
            "prefix": "186.96.174.0/24"
        },
        "199.36.240.0/22": {
            "attr": {
                "1": 0,
                "2": [
                    [
                        2,
                        [
                            27418,
                            8100,
                            174,
                            13238
```

(continues on next page)

(continued from previous page)


```
                        ]
                    ]
                ],
                "3": "10.75.44.254",
                "5": 100,
                "7": [
                    13238,
                    "100.43.92.159"
                ],
                "8": [
                    "174:21001",
                    "174:22013",
                    "8100:50",
                    "29761:50"
                ],
                "9": "172.17.0.7",
                "10": [
                    "2.2.2.2"
                ]
            },
            "prefix": "199.36.240.0/22"
        }
    },
    "status": true
}
```


## 1.6 Write Customized YABGP through Extension Handlers

YABGP is an extendable BGP probe which can be used to establish BGP session with all kinds of routers, and send/received BGP messages to/from them. The decoded BGP messages with `json` format will be written in a file by default. But through our `handler mechanism`, you can writed your own bgpd process by implementation of your own handler.

What you only need to do in your own hander is to deside how to process all kinds of receveid BGP messages and BGP session connection situation. The handler will inherit from `BaseHandler` and implement all methods of it. you can wirte your own code in each message process method and do what to want to do with received messages like maintain RIB, insert into database, etc. If you want to add more configurable options, please reference the `DefaultHandler`.

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

from __future__ import print_function
import sys

from yabgp.agent import prepare_service
from yabgp.handler import BaseHandler


class CliHandler(BaseHandler):
    def __init__(self):
        super(CliHandler, self).__init__()

    def init(self):
        pass
```

(continues on next page)

(continued from previous page)

```
    def on_update_error(self, peer, timestamp, msg):
        print('[-] UPDATE ERROR,', msg)

    def route_refresh_received(self, peer, msg, msg_type):
        print('[+] ROUTE_REFRESH received,', msg)

    def keepalive_received(self, peer, timestamp):
        print('[+] KEEPALIVE received')

    def open_received(self, peer, timestamp, result):
        print('[+] OPEN received,', result)

    def update_received(self, peer, timestamp, msg):
        print('[+] UPDATE received,', msg)

    def notification_received(self, peer, msg):
        print('[-] NOTIFICATION received,', msg)

    def on_connection_lost(self, peer):
        print('[-] CONNECTION lost')

    def on_connection_failed(self, peer, msg):
        print('[-] CONNECTION failed,', msg)

    def on_established(self, peer, msg):
        print('[-] ESTABLISHED,', msg)

def main():
    try:
        cli_handler = CliHandler()
        prepare_service(handler=cli_handler)
    except Exception as e:
        print(e)


if __name__ == '__main__':
    sys.exit(main())
```

How to run it? very simple! let's call this file as `my_bgpd.py`, and you can run it just like `yabgpd`

**Note:** Please make sure you have install yabgp for requirements, you can do that through `pip install yabgp`

```
$ python my_bgpd.py  --bgp-local_as=100 --bgp-remote_addr=1.1.1.1 --bgp-remote_as=100␣
↪--bgp-afi_safi=ipv4,bgpls,flowspec
```

## 1.7 Tools

Here are some tools can be used for yabgp.

### 1.7.1 Route Injector

Tools location: https://github.com/smartbgp/yabgp/blob/master/tools/route_injector

This tool can be used to send bgp update messages to yabgp process. For example, you start a yabgp process:

```
$ python yabgp/bin/yabgpd --bgp-local_as=100 --bgp-local_addr=127.0.0.1 \
                          --bgp-remote_addr=2.2.2.2 --bgp-remote_as=100
```

If you have some BGP messages come from some other yabgp process, like:

```
$ pwd
/home/yabgp/data/bgp/1.1.1.1/msg
$ ls
1450668274.82.msg 1450668593.59.msg
```

We want to send all the BGP message received from peer `1.1.1.1` to `2.2.2.2`. We can use route injector like this:

```
$ python route_injector --rest-host=127.0.0.1 --rest-port=8801  \
                        --message-json=/home/yabgp/data/bgp/1.1.1.1/msg/1450668274.82.
↪msg \
                        --peerip=2.2.2.2
Percent: [########################################          ] 81.05%
```

Then, route-injector will read bgp message file and try to send all bgp messages to peer `2.2.2.2` through REST API. When finised:

```
Percent: [##################################################] 100.00%
Total messages:   128444.
Success send out: 15109
Failed send out:  113335
```

### 1.7.2 Postman Collection

Located in `/yabgp/tools/Yabgp.postman_collection.json`. You can import this collection into POSTMAN(http://www.getpostman.com/) and there are some REST API request examples.

## 1.8 Reference

### 1.8.1 RFCs

- RFC1997(BGP Communities Attribute)
- RFC2385(Protection of BGP Sessions via the TCP MD5 Signature Option)
- RFC2439(BGP Route Flap Damping)
- RFC2545(Use of BGP-4 Multiprotocol Extensions for IPv6 Inter-Domain Routing)
- RFC2858(Multiprotocol Extensions for BGP-4)
- RFC2918(Route Refresh Capability for BGP-4)
- RFC3031(Multiprotocol Label Switching Architecture)
- RFC3032(MPLS Label Stack Encoding)

- RFC3065 (Autonomous System Confederations for BGP)
- RFC3107(Carrying Label Information in BGP-4)
- RFC3392(Capabilities Advertisement with BGP-4)
- RFC4271(A Border Gateway Protocol 4)
- RFC4360(BGP Extended Communities Attribute)
- RFC4364(BGPMPLS IP Virtual Private Networks (VPNs))
- RFC4456(BGP Route Reflection An alternative to Full Mesh Internal BGP)
- RFC4724(Graceful Restart Mechanism for BGP)
- RFC4760( Multiprotocol Extensions for BGP-4)
- RFC4798(Connecting IPv6 Islands over IPv4 MPLS Using IPv6 Provider Edge Routers (6PE))
- RFC4893(BGP Support for Four-octet AS Number Space).txt
- RFC5065(Autonomous System Confederations for BGP)
- RFC5291(Outbound Route Filtering Capability for BGP-4)
- RFC5396(Textual Representation of Autonomous System (AS) Numbers).txt
- RFC5492(Capabilities Advertisement with BGP-4)
- RFC5668(4-Octet AS Specific BGP Extended Community)
- RFC5701(IPv6 Address Specific BGP Extended Community Attribute)
- RFC6368(Internal BGP as the ProviderCustomer Edge Protocol for BGPMPLS IP Virtual Private Networks (VPNs))
- RFC6472 (Recommendation for Not Using AS_SET and AS_CONFED_SET in BGP)
- RFC6513(Multicast in MPLS BGP IP VPNs)
- RFC6774(Distribution of Diverse BGP Paths)
- RFC7999 (BLACKHOLE Community)

### 1.8.2 Code

### 1.8.3 IOS/IOX

1. BGP FlowSpec configuration

```
class-map type traffic match-all bflow
 match destination-address ipv4 2.2.2.0 255.255.255.0
 match source-address ipv4 3.3.0.0 255.255.0.0
 match protocol ipv4 gre eigrp icmp igmp ospf pim
 match source-port 80 8080 8081 8082 8083
 match ipv4 icmp-type 2 3 5 6
 match ipv4 icmp-code 2
 match tcp-flag 40
 match packet length 254 254-300
 match dscp cs5 cs6
 match fragment-type  dont-fragment
 match destination-port 8080-8090
 end-class-map
```

(continues on next page)

(continued from previous page)

```
policy-map type pbr bflow
class type traffic bflow
 set dscp ef

router bgp 100
bgp router-id 1.1.1.1
address-family ipv4 unicast
 network 1.1.1.1/32
!
address-family vpnv4 unicast
!
address-family ipv6 unicast
!
address-family ipv4 flowspec
!
neighbor 10.75.44.121
 remote-as 100
 password encrypted 121A0C041104
 update-source MgmtEth0/0/CPU0/0
 address-family ipv4 flowspec
  route-policy XR in
  route-policy XR out
 !
!


flowspec
address-family ipv4
 service-policy type pbr bflow
 service-policy type pbr match_dest_110.1.1.x_drop
 service-policy type pbr match_src_10.1.1.10_police
 service-policy type pbr match_proto_gre_redir_nh_ipv4
!
```

CHAPTER 2

# Support

There are many jobs need to do in future. We are working hardly on that. So any of your ideas is welcome.

Please use GitHub issue system or submit pull request.

CHAPTER 3

# Indices and tables

- genindex
- modindex
- search