Tutorial: Using GoBGP as an IXP connecting router

Shu Sugimoto

JPNAP / INTERNET MULTIFEED CO.

IX.br Forum 10

2016/12/07(Wed)

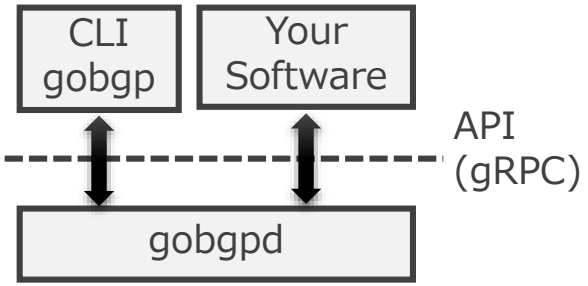
About this presentation

- Show you how GoBGP can be used as a software router in conjunction with quagga
- (Tutorial) Walk through the setup of IXP connecting router using GoBGP

- This is going be/was spoken at IX.br Forum 10
- Slides available at SlideShare

GoBGP

- New BGP implementation
- OSS developed by NTT Lab SIC
  - https://github.com/osrg/gobgp
- Written in Golang
- High performance
  - exploits multi-cores natively
- Automation friendly
  - API first principle
    - CLI on top of API



Key features 1/2

- full-featured CLI
- Multiprotocol support
  - IPv4, IPv6, Labeled IPv4/IPv6, VPN IPv4/IPv6, EVPN, flowspec IPv4/IPv6/L2
- Flexible Policy
- Graceful Restart
  - both restarting/helper speaker role
- Route Reflector
- Route Server

Key features 2/2

- MRT dumping
- BMP
- RPKI validation
- FIB manipulation
- gRPC API
- Standard configuration format
  - structured based on OpenConfig
  - supports toml/yaml/json/hcl

Components

- **gobgpd**
  - main daemon process which implements BGP
  - can be controlled via gRPC API
  - configuration file also supported
- **gobgp**
  - full-featured CLI
  - convert human friendly commands into gRPC API call
    - and vice versa
- **configuration file** (optional)
  - popular way to define the behavior of gobgpd
  - written in toml/yaml/json/hcl

gobgp CLI command example

- show list of neighbors

```
$ gobgp neighbor
Peer              AS     Up/Down State       |#Received  Accepted
10.1.0.101     65001 3d 08:25:02 Establ      |        1         1
10.173.176.103 65003 3d 08:25:00 Establ      |        1         1
10.173.176.211 64686       never Active      |        0         0
```

- show RIB

```
$ gobgp global rib
    Network             Next Hop             AS_PATH              Age
Attrs
*>  10.1.0.0/16         0.0.0.0                                   3d
08:06:32 [{Origin: i} {Med: 0}]
N*> 10.3.0.0/16         10.173.176.103       65003                3d
08:34:01 [{Origin: i} {Communities: 65001:1000}]
N*  10.3.0.0/16         10.173.176.103       65003
00:00:03   [{Origin: i} {Communities: 65001:1000}]
N*> 10.4.0.0/16         10.1.14.104          65004                3d
08:34:03 [{Origin: i} {LocalPref: 100}]
```

gobgp CLI command example

- neighbor operations

```
### perform peer softresetin
$ gobgp neighbor 10.1.0.101 softresetin

### perform peer reset
$ gobgp neighbor 10.1.0.101 reset

### perform peer disable (shutdown)
$ gobgp neighbor 10.1.0.101 disable

### perform peer enable
$ gobgp neighbor 10.1.0.101 enable
```

gobgp CLI command example

- You can even add/delete peers/routes/policies on the fly
  - Some easy use case can be accomplished without using configuration file

```
### launch gobgpd (need privilege to listen on tcp 179)
$ sudo gobgpd

### (open new terminal)
### set AS and router-id
$ gobgp global as 1 router-id 1.1.1.1

### add neighbor
$ gobgp neighbor add 192.0.2.2 as 2

### add route into RIB, which will then advertised to peers
$ gobgp global rib add -a ipv4 10.0.0.0/24 med 10 community 100:100
```

gobgp CLI command example

- `-j` support
  - Every command supports json output

```
$ gobgp global rib 10.4.0.0/16 -j
{"10.4.0.0/16":[{"nlri":{"prefix":"10.4.0.0/16"},"attrs":[{"type":1,"val
ue":0},{"type":2,"as_paths":[{"segment_type":2,"num":1,"asns":[65004]}]}
,{"type":3,"nexthop":"10.1.14.104"},{"type":5,"value":100},{"type":8,"co
mmunities":[4259907539]}],"age":1480845275,"validation":"not-
found","source-id":"10.1.0.101","neighbor-ip":"10.1.0.101"}]}

### pretty print using python
$ gobgp global rib 10.4.0.0/16 -j | python -mjson.tool
{
    "10.4.0.0/16": [
        {
            "age": 1480845275,
            "attrs": [
                {
                    "type": 1,
 : (snip)
```

gobgp CLI command example

- Event monitoring
  - The data will be sent from gobgpd through the gRPC connection channel when events occur
    - Push notification
  - One implementation of the event driven application

```
$ gobgp monitor global rib
[ROUTE] 10.3.0.0/16 via 10.173.176.103 aspath [65003] attrs [{Origin: i}
{Communities: 65001:1000, 65001:2003}]
[ROUTE] 10.4.0.0/16 via 10.1.14.104 aspath [65004] attrs [{Origin: i}
{LocalPref: 100} {Communities: 65001:2003}]
[DELROUTE] 10.3.0.0/16 via 10.173.176.103 aspath [65003] attrs [{Origin:
i} {Communities: 65001:1000, 65001:2003}]
[ROUTE] 10.3.0.0/16 via 10.173.176.103 aspath [65003] attrs [{Origin: i}
{Communities: 65001:1000, 65001:2003}]
[ROUTE] 10.3.0.0/16 via 10.173.176.103 aspath [65003] attrs [{Origin: i}
{Communities: 65001:1000, 65001:2003}]
(waiting for further events…)
```

Who use GoBGP?

- **IXP Route Server**
  - JPNAP
- **Monitoring compoment**
  - FastNetMon
    - DoS/DDoS analyzer
  - BGPmon
    - BGP routing information monitor
  - Cloudwatt
    - is an OpenStack based public cloud service
    - They seems to be using it as a component of Looking Glass

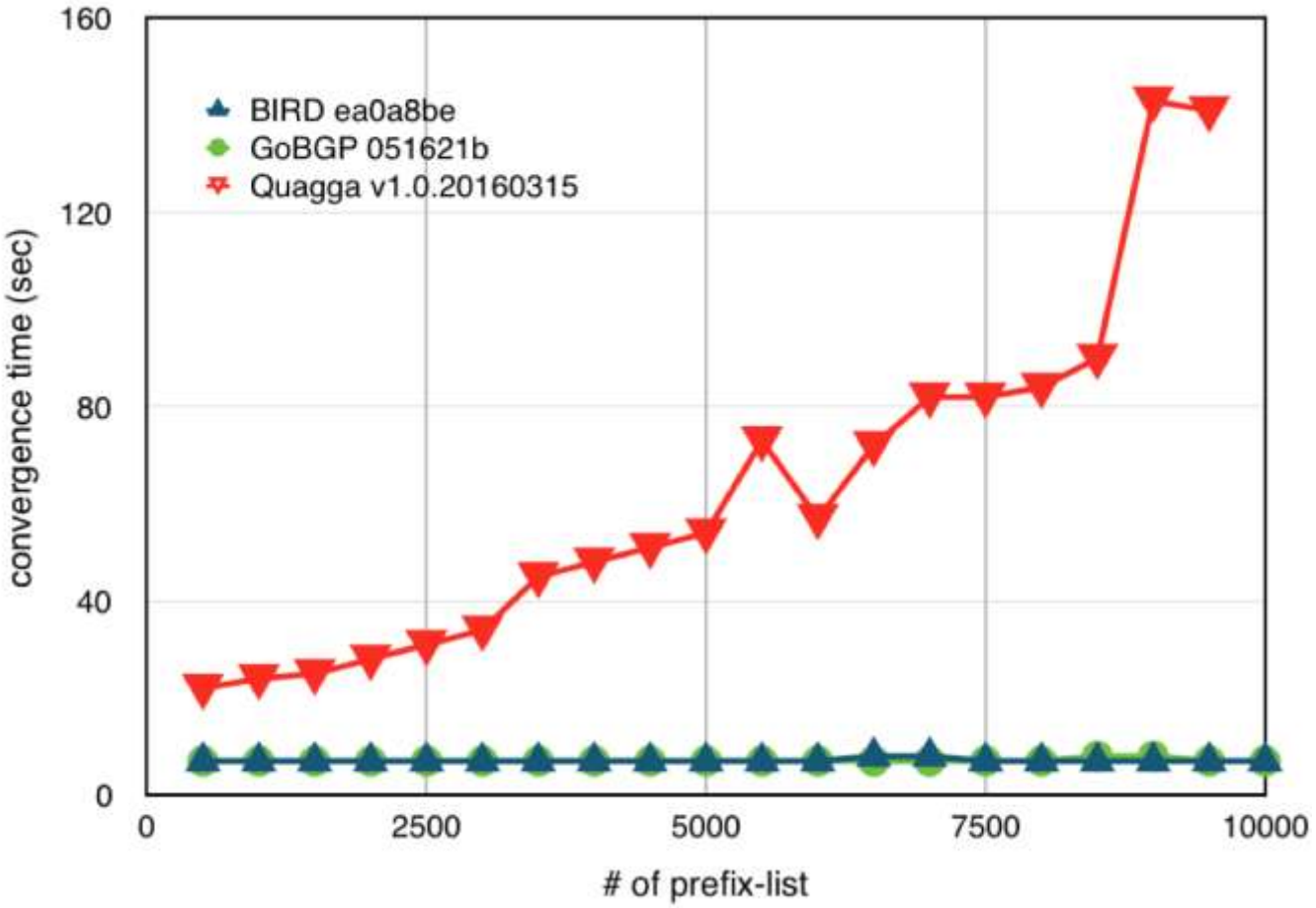
Who use GoBGP?

- **SDN solution component**
  - Project Calico
  - Contiv netplugin
    - Both are network plugin for containers
- **Test virtualization**
  - ex.) One is using GoBGP to virtualize the Lab environment for testing BGP routing policies
    - Only test target is a real router, everything else is VM
    - GoBGP is used as a route originator
      - gRPC API is used to generate arbitrary routes
    - http://www.slideshare.net/ssuser6a8d29/gobgp
      - in Japanese :P

Why use GoBGP?

- **Automation Friendliness**
  - It's definitely easier to implement whatever comes up in your mind
    - Everything is exposed through API
  - `gobgp -j` can be a good start point of "thinking about automation"
    - can skip the painful "output parsing" part, which broke a lot of people's motivation
    - much easier to begin with for operators ☺
- **Performance**
  - Especially in larger deploy

GoBGP performance



Quagga won’t scale if the policy is huge

Comparing the convergence time in Y-axis (between beginning of first peer up and the last update sent)
X-axis = # of IPs in prefix list / fixed # of peers = 100, number of prefixes per peer = 100, Route Server setup
Graph generated using bgperf (https://github.com/osrg/bgperf), in Jan 2016

Why not? Then…

- “Because it’s not stable yet, isn’t it?”
- “Because no one is using it as yet, right?”
- (many many many reasons…)

Why not? Then…

- I can't say 100% "Yes it's stable" but…
  - We use it in production as a Route Server at JPNAP and so far not facing any big issue
  - Development team are amazingly fast
    - in response, finding causes, and providing patches
- I really would like some of you to consider trying GoBGP
  - as a software router
    - There's no one still AFAIK
  - Need help? Find bug? Anything you want to discuss?
    - Open an issue at GitHub
    - or you can join open slack channel
      - https://slackin-gobgp.mybluemix.net/

Bugs resolved

- Here are the list of bugs found and fixed while I was working on making this tutorial…
  - server: fix bug of deleteNeighbor() #1184
    - https://github.com/osrg/gobgp/pull/1184
  - zebra: add flags for recursive nexthop lookup if necessary #1179
    - https://github.com/osrg/gobgp/pull/1179
  - fix several bugs related to rpki and policy #1178
    - https://github.com/osrg/gobgp/pull/1178
  - gobgpd: support global policy assignment update via configuration file #1177
    - https://github.com/osrg/gobgp/pull/1177
- Some other bugs still under discussion
- Many thanks to @wataru and @tomo

FIB manipulation

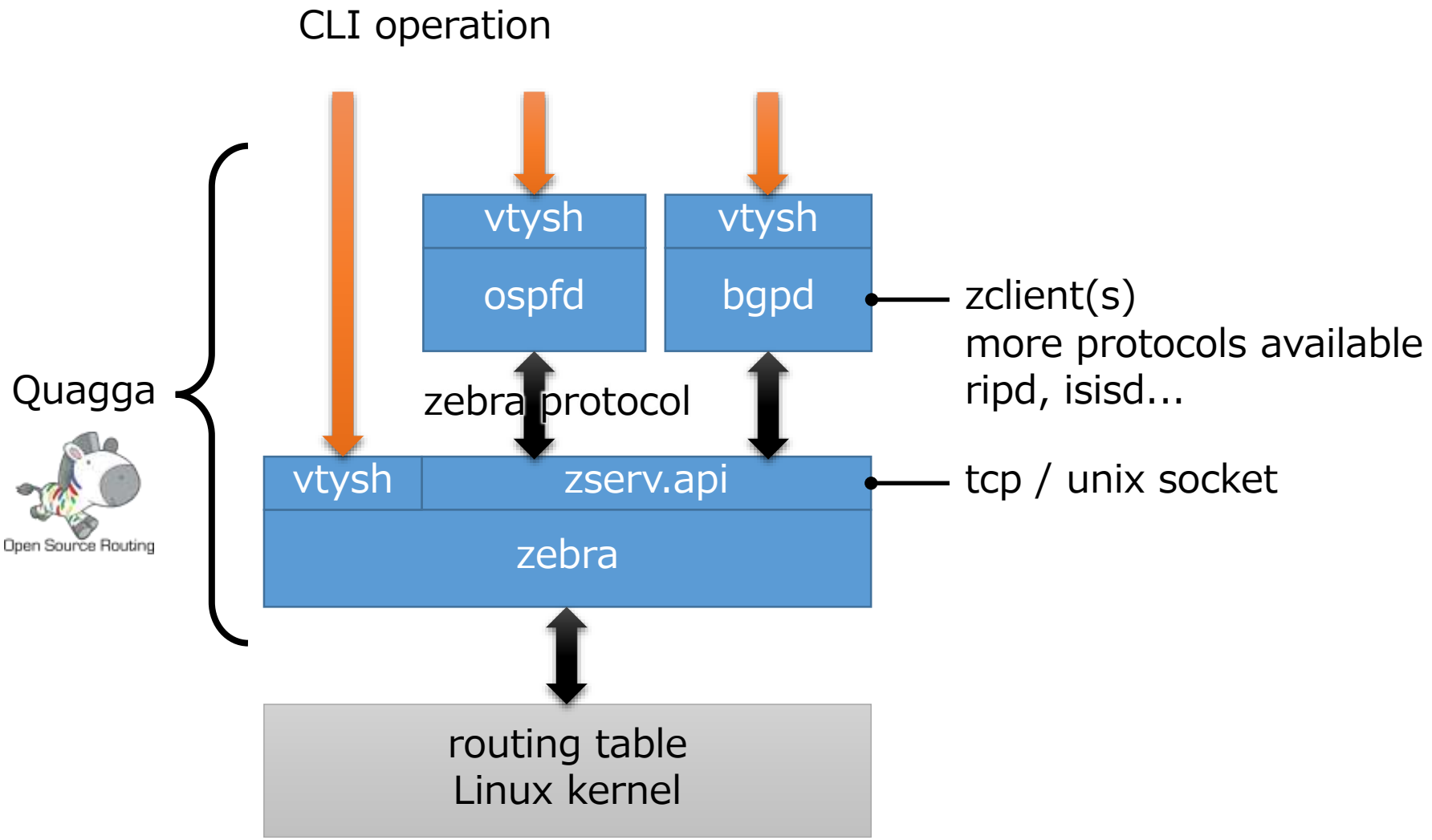
- GoBGP is just a bgp daemon and itself does not contain any functionality to modify routing table
- If you like to use GoBGP as a component of software router and do packet forwarding, you need to implement that
- There are two options to achieve FIB manipulation with GoBGP
  - Use built-in zebra integration
  - Write your own code using gRPC API
- In this tutorial I'll introduce zebra integration and show you how we can use it

cf.) GoPlane

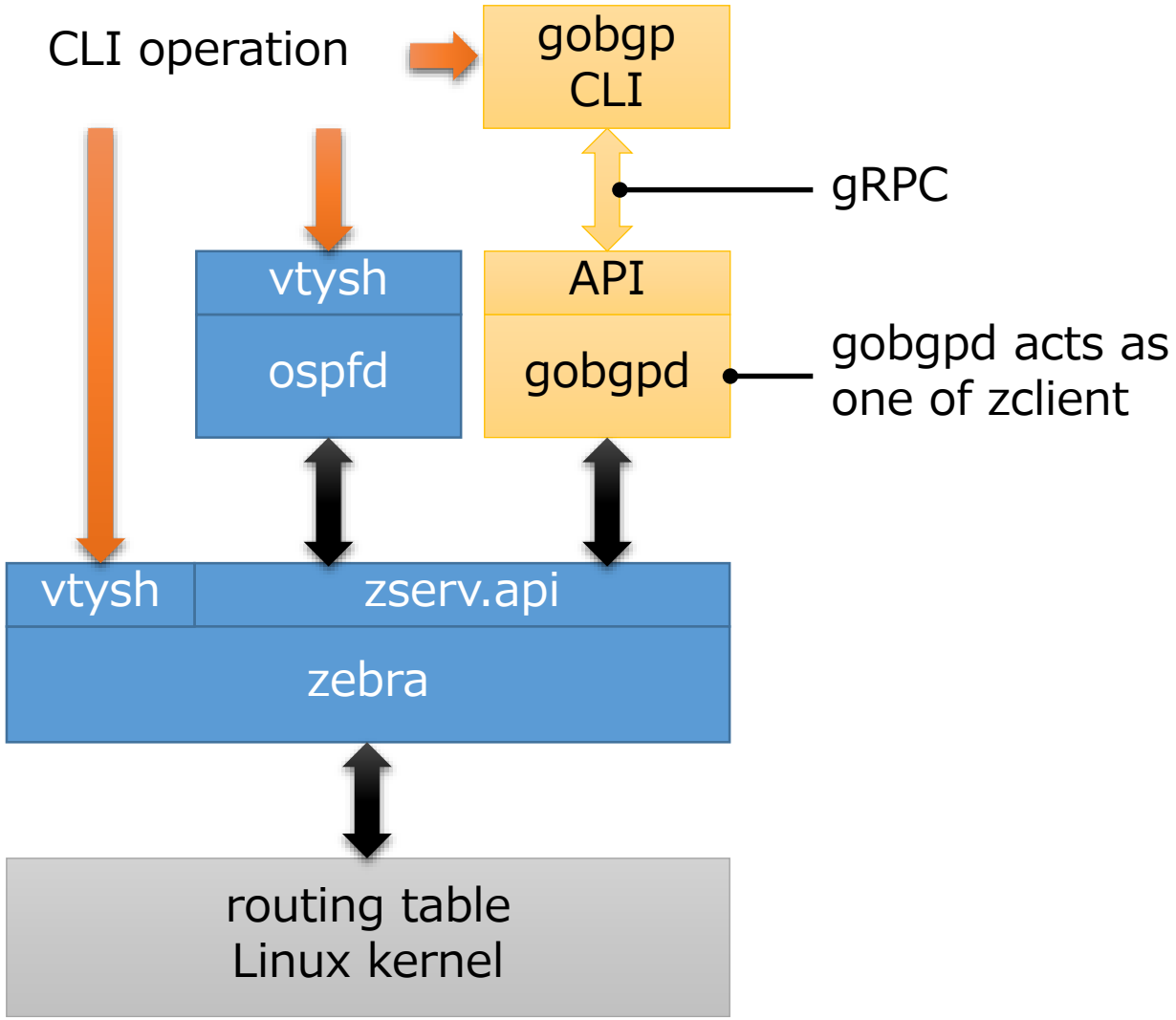
- https://github.com/osrg/goplane
- Utilize gobgpd gRPC API and perform things like FIB manipulation on Linux platform
  - Can create EVPN/VxLAN fabric
- Also can modify iptables rules based on received FlowSpec routes

- Can’t do any complex routing
  - ex.) Recursive next-hop resolving not supported
  - Not suitable for the use case in this tutorial

Quagga architecture



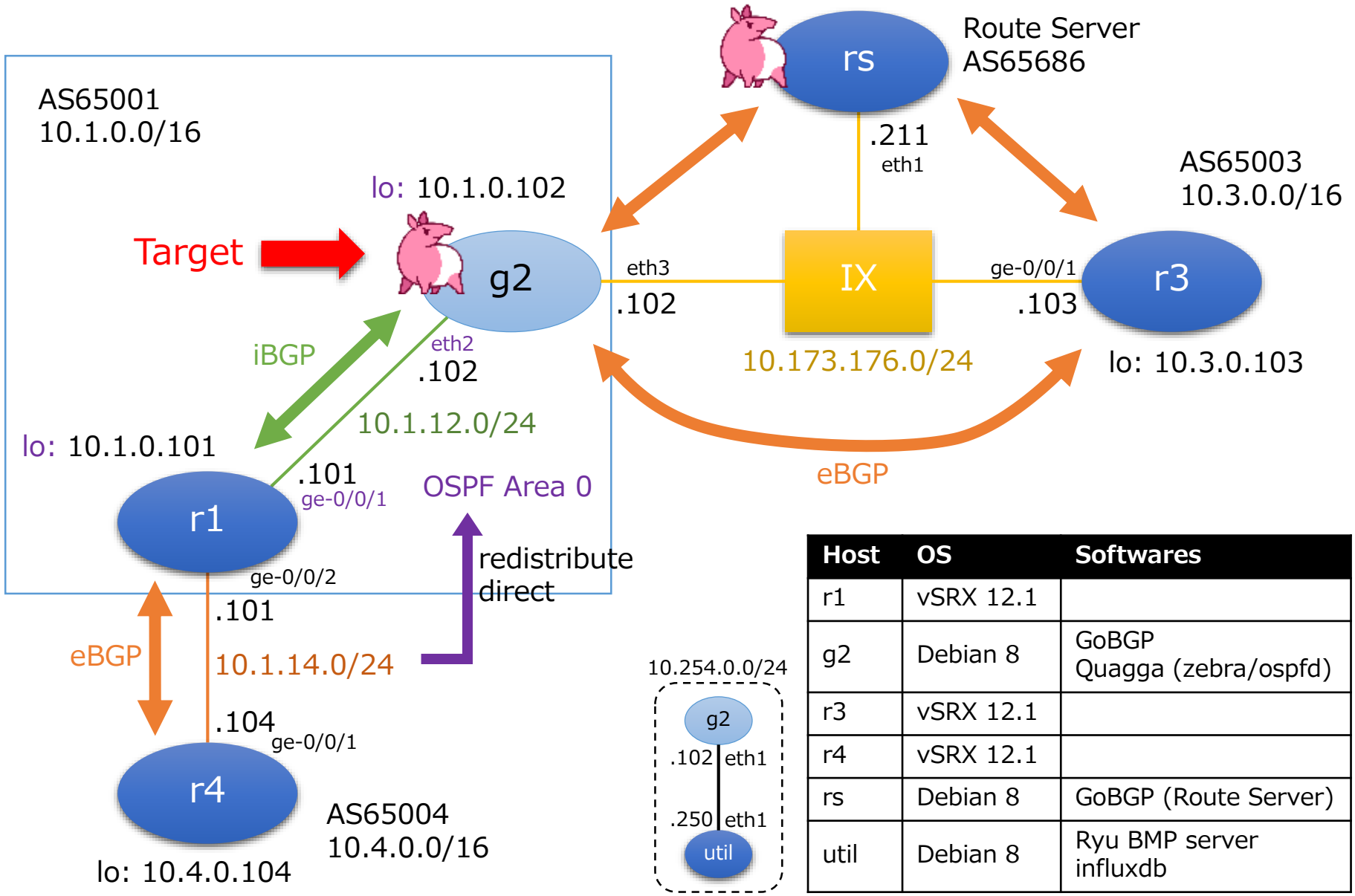
Zebra integration of GoBGP



Tutorial

- Target: IXP connecting router
- You will walk through
  - Installation of GoBGP
  - gobgpd administration via systemd
  - Writing configuration file
  - Adding eBGP/iBGP peers
  - Applying policy
    - Including example use case to control route advertising over Route Server
  - FIB manipulation with zebra integration
  - RPKI setup
  - MRT/BMP setup
  - InfluxDB integration setup

Topology



Host	OS	Softwares
r1	vSRX 12.1	
g2	Debian 8	GoBGP Quagga (zebra/ospfd)
r3	vSRX 12.1	
r4	vSRX 12.1	
rs	Debian 8	GoBGP (Route Server)
util	Debian 8	Ryu BMP server influxdb

gobgp-tutorial

- You can instantly build the demo topology using Vagrant
- Vagrantfile available at GitHub
  - https://github.com/s2ugimot/gobgp-tutorial
- Follow the instructions in README.md
  - Everything except the GoBGP in **g2** will be set up

Demo setup

- MacBook Pro 13r (Early 2015)
  - Mac OS X 10.11.6 (El Capitan)
  - 3.1GHz Intel Core i7
  - 16GB RAM (at least > 8GB)
- Vagrant 1.8.1
  - vagrant-host-shell 0.0.4
  - vagrant-junos 0.2.1
- VirtualBox 5.0.28

- Should work on Windows/Linux too
  - haven’t tested though :P

System consideration in real world

- RAM
  - > 16GB
  - Recommend 32GB or more if you handle IPv4 full routes = 600k routes with multiple eBGP peers
  - This is by design
    - No extensive tweaks, keep it simple, just buy memory
- CPU
  - > 2cores
  - The more, the better performance
- VM is fine
  - If forwarding is not the issue

Tutorial: Step by step

Install Go

- Just follow the instruction on official web
  - https://golang.org/doc/install
  - It’s quite simple, just extract tar.gz and add to $PATH
  - Choose go1.5 or above

Build GoBGP

```
### build binaries
g2 $ go get github.com/osrg/gobgp/gobgpd
g2 $ go get github.com/osrg/gobgp/gobgp

### copy them to somewhere under $PATH
g2 $ cp $GOPATH/bin/* /usr/local/sbin

### optional: install shell completion for gobgp command
g2 $ cp $GOPATH/src/github.com/osrg/gobgp/tools/completion/*.bash
/etc/bash_completion.d/
```

- Built binaries are portable
  - Libraries are statically linked into binary
- Should preserve the $GOPATH
  - To make it reproducible
  - Go itself does not provide any good solution still
  - ex.) build in container and keep the image

Setup systemd unit file

- Prepare systemd unit file for gobgpd process to let it managed by systemd

Setup systemd unit file

```
### create a unit file for gobgpd
g2 $ cat << EOF > /etc/systemd/system/gobgpd.service
[Unit]
Description=gobgpd
After=network.target syslog.target

[Service]
Type=simple
PermissionsStartOnly=yes
User=quagga
ExecStartPre=/sbin/setcap 'cap_net_bind_service=+ep'
/usr/local/sbin/gobgpd
ExecStart=/usr/local/sbin/gobgpd -f /etc/gobgp/gobgpd.conf -t yaml --
cpus=2
ExecReload=/bin/kill -s HUP $MAINPID
ExecStop=/bin/kill -s TERM $MAINPID

[Install]
WantedBy=multi-user.target
EOF

g2 $ systemctl daemon-reload
```

Setup systemd unit file

- Set process user same as that of quagga service
  - because gobgpd needs rw permission to zserv.api to speak to zebra
- Use `setcap` to let unprivileged user listen on port < 1024
- Security concern: you should not run any important gobgpd on shared servers!
  - Anyone has access to full-control through gRPC API
    - At least you should block access to API port (default tcp 50051) by applying iptables rules from other hosts
  - Exposing `setcap`-ed binary to others might compromise your server seucrity

Adjusting options

```
Application Options:
  -f, --config-file=      specifying a config file
  -t, --config-type=      specifying config type (toml, yaml, json)
(default: toml)
  -l, --log-level=        specifying log level
  -p, --log-plain         use plain format for logging (json by default)
  -s, --syslog=           use syslogd
      --syslog-facility=  specify syslog facility
      --disable-stdlog    disable standard logging
      --cpus=             specify the number of CPUs to be used
      --api-hosts=        specify the hosts that gobgpd listens on
(default: :50051)
  -r, --graceful-restart  flag restart-state in graceful-restart
capability
  -d, --dry-run           check configuration
      --pprof-host=       specify the host that gobgpd listens on for
pprof (default: localhost:6060)
      --pprof-disable     disable pprof profiling
```

- Update `ExecStart` section in unit file to fit your needs
- You can specify how many cores you want to use

Create the first configuration

```
global:
  config:
    as: 65001
    router-id: 10.1.0.102
```

- Minimal configuration includes AS and router-id

- gobgpd supports toml/yaml/json/hcl format
- I would recommend using toml
  - used in official document
- Here I will use yaml through this tutorial
  - It’s easier to read

toml configuration example

```
[global.config]
  as = "65001"
  router-id = "10.1.0.102”

[[neighbors]]
  [neighbors.config]
    neighbor-address = 10.173.176.103
    peer-as = 65003

[[neighbors]]
  [neighbors.config]
    neighbor-address = 10.1.0.101
    peer-as = 65001
  [neighbors.transport.config]
    local-address = 10.1.0.102
```

- toml is better in two reasons
  - It is easy to “copy & paste” config fragments
  - Element ordering does not matter
    - ex.) You can define a neighbor, a policy for it, then other neighbor
    - Arbitrary ordering helps you organize the configuration

Launch gobgpd

```
g2 $ systemctl start gobgpd
g2 $ systemctl status gobgpd
● gobgpd.service - gobgpd
   Loaded: loaded (/etc/systemd/system/gobgpd.service; disabled)
   Active: active (running) since Thu 2016-12-01 05:56:17 UTC; 7s ago
  Process: 5987 ExecStop=/bin/kill -s TERM $MAINPID (code=exited,
status=0/SUCCESS)
  Process: 5981 ExecReload=/bin/kill -s HUP $MAINPID (code=exited,
status=0/SUCCESS)
  Process: 6057 ExecStartPre=/sbin/setcap cap_net_bind_service=+ep
/usr/local/sbin/gobgpd (code=exited, status=0/SUCCESS)
 Main PID: 6061 (gobgpd)
   CGroup: /system.slice/gobgpd.service
           └─6061 /usr/local/sbin/gobgpd -f /etc/gobgp/gobgpd.conf -t
yaml --cpus=2

Dec 01 05:56:17 g2 gobgpd[6061]: {"level":"info","msg":"gobgpd
started","time":"2016-12-01T05:56:17Z"}
Dec 01 05:56:17 g2 gobgpd[6061]:
{"Topic":"Config","level":"info","msg":"Finished reading the config
file","time":"2016-12-01T05:56:17Z"}
```

Logging

- Default to output in json format
  - easy for computers
  - (a bit) hard for humans :P
  - You can switch to plain text mode by passing `-p/--log-plain`
- All logs will be shown in stdout
  - can be disabled by `--disable-stdlog`
- Syslog is also supported
  - exactly the same log will be produced

Logging

```
### check log via journald
g2 $ journalctl -xn -f -u gobgpd
-- Logs begin at Mon 2016-12-05 05:05:34 UTC. --
Dec 05 05:21:57 g2 gobgpd[1352]: {"level":"info","msg":"gobgpd
started","time":"2016-12-05T05:21:57Z"}
Dec 05 05:21:57 g2 gobgpd[1352]:
{"Topic":"Config","level":"info","msg":"Finished reading the config
file","time":"2016-12-05T05:21:57Z"}

### check log via file (via rsyslogd through systemd/journald)
g2 $ tailf /var/log/syslog
Dec  5 05:21:57 g2 gobgpd[1352]: {"level":"info","msg":"gobgpd
started","time":"2016-12-05T05:21:57Z"}
Dec  5 05:21:57 g2 gobgpd[1352]:
{"Topic":"Config","level":"info","msg":"Finished reading the config
file","time":"2016-12-05T05:21:57Z"}
```

Configure first eBGP peer

- Establish eBGP peer between **g2** and **r3**
- Receive routes from **r3**

- Apply policy to routes received from **r3**
  - **r3** is a peer over IXP
  - will `tag` to identify the routes received from IXP peers using community
    - Use ”65000:1000”

Configure first eBGP peer

```
global:
  config:
    as: 65001
    router-id: 10.1.0.102

neighbors:
  - config:
      neighbor-address: 10.173.176.103
      peer-as: 65003
```

```
g2 $ systemctl reload gobgpd
```

- neighbor/policy configuration can be dynamically applied by sending SIGHUP to the gobgpd process
  - We define `reload` to send SIGHUP in systemd unit file
- If there is a syntax error in configuration file, gobgpd just abort reloading and remain on the current state
  - You can see exact location causing error in log

Configure first eBGP peer

```
g2 $ gobgp neigh
Peer              AS Up/Down State       |#Received  Accepted
10.173.176.103 65003   never Active      |        0          0

### wait for a while...
g2 $ gobgp neigh
Peer              AS  Up/Down State       |#Received  Accepted
10.173.176.103 65003 00:00:36 Establ      |        1          1
```

Configure first eBGP peer

```
g2 $ gobgp neighbor 10.173.176.103
BGP neighbor is 10.173.176.103, remote AS 65003
  BGP version 4, remote router ID 10.3.0.103
  BGP state = established, up for 00:03:43
  BGP OutQ = 0, Flops = 0
  Hold time is 90, keepalive interval is 30 seconds
  Configured hold time is 90, keepalive interval is 30 seconds
  Neighbor capabilities:
    multiprotocol:
        ipv4-unicast:   advertised and received
    route-refresh:      advertised and received
    graceful-restart:   received
    4-octet-as: advertised and received
    cisco-route-refresh:        received
  Message statistics:
                        Sent       Rcvd
    Opens:                 1          1
    Notifications:         0          0
    Updates:               0          1
    Keepalives:            8         10
 : (snip)
```

Configure first eBGP peer

```
### check received routes
### junos: show route receiving-protocol bgp 10.173.176.103
g2 $ gobgp neighbor 10.173.176.103 adj-in
    Network              Next Hop             AS_PATH              Age
Attrs
    10.3.0.0/16          10.173.176.103       65003
00:12:08   [{Origin: i}]

### check BGP RIB
### junos: show route protocol bgp
g2 $ gobgp global rib
gobgp global rib
    Network              Next Hop             AS_PATH              Age
Attrs
*>  10.3.0.0/16          10.173.176.103       65003
00:17:34   [{Origin: i}]
```

- Caveats: you need to use `-j` option and see in json format to see more detailed attributes like router-id or source (from which neighbor the route has received)
  - Feel free to open an issue at GitHub ☺

Configure first eBGP peer

```
### see detailed information using `-j`
g2 $ gobgp global rib -j | python -mjson.tool
{
    "10.3.0.0/16": [
        {
            "age": 1480923494,
            "attrs": [
             : (snip)
            ],
            "neighbor-ip": "10.173.176.103",
            "nlri": {
                "prefix": "10.3.0.0/16"
            },
            "source-id": "10.3.0.103"
        }
    ]
}
```

Apply policy to received routes

```
policy-definitions:
  - name: tag-ixp-neighbors
    statements:
      - conditions:
          match-neighbor-set:
            neighbor-set: ixp-neighbors
            match-set-options: any
        actions:
          bgp-actions:
            set-community:
              options: add
              set-community-method:
                communities-list:
                  - "65001:1000"

defined-sets:
  neighbor-sets:
    - neighbor-set-name: ixp-neighbors
      neighbor-info-list:
        - 10.173.176.103
```

Apply policy to received routes

```
global:
  config:
    as: 65001
    router-id: 10.1.0.102
  apply-policy:
    config:
      import-policy-list:
        - tag-ixp-neighbors
      default-import-policy: accept-route
```

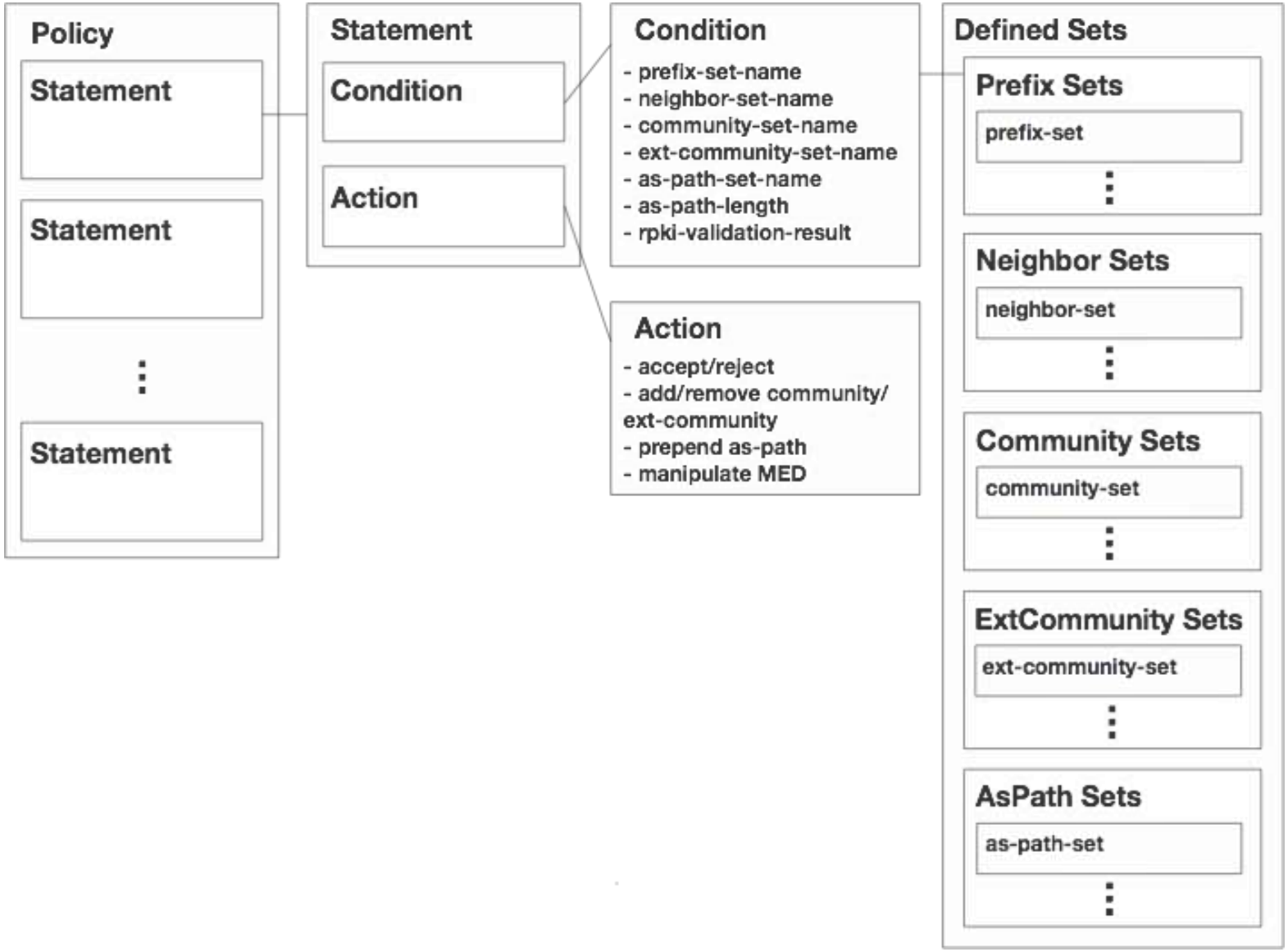
```
### apply policy
g2 $ systemctl reload gobgpd

### see what happens
g2 $ gobgp global rib
    Network              Next Hop             AS_PATH              Age
Attrs
*>  10.3.0.0/16         10.173.176.103       65003
01:43:37   [{Origin: i} {Communities: 65001:1000}]
```

Policy definition structure

- Policy consists of statements
- Each statement has condition(s) and action(s)
  - condition specifies the state of NLRIs to match
    - ex.) prefix, neighbor, AS_PATH, community, …
  - actions specifies what to do with the NLRIs
    - accept / reject
    - modify path attributes
      - community, MED, local-pref, AS_PATH, next-hop
- Some condition refers to defined-sets
  - ex.) prefix-set, neighbor-set, community-set, …

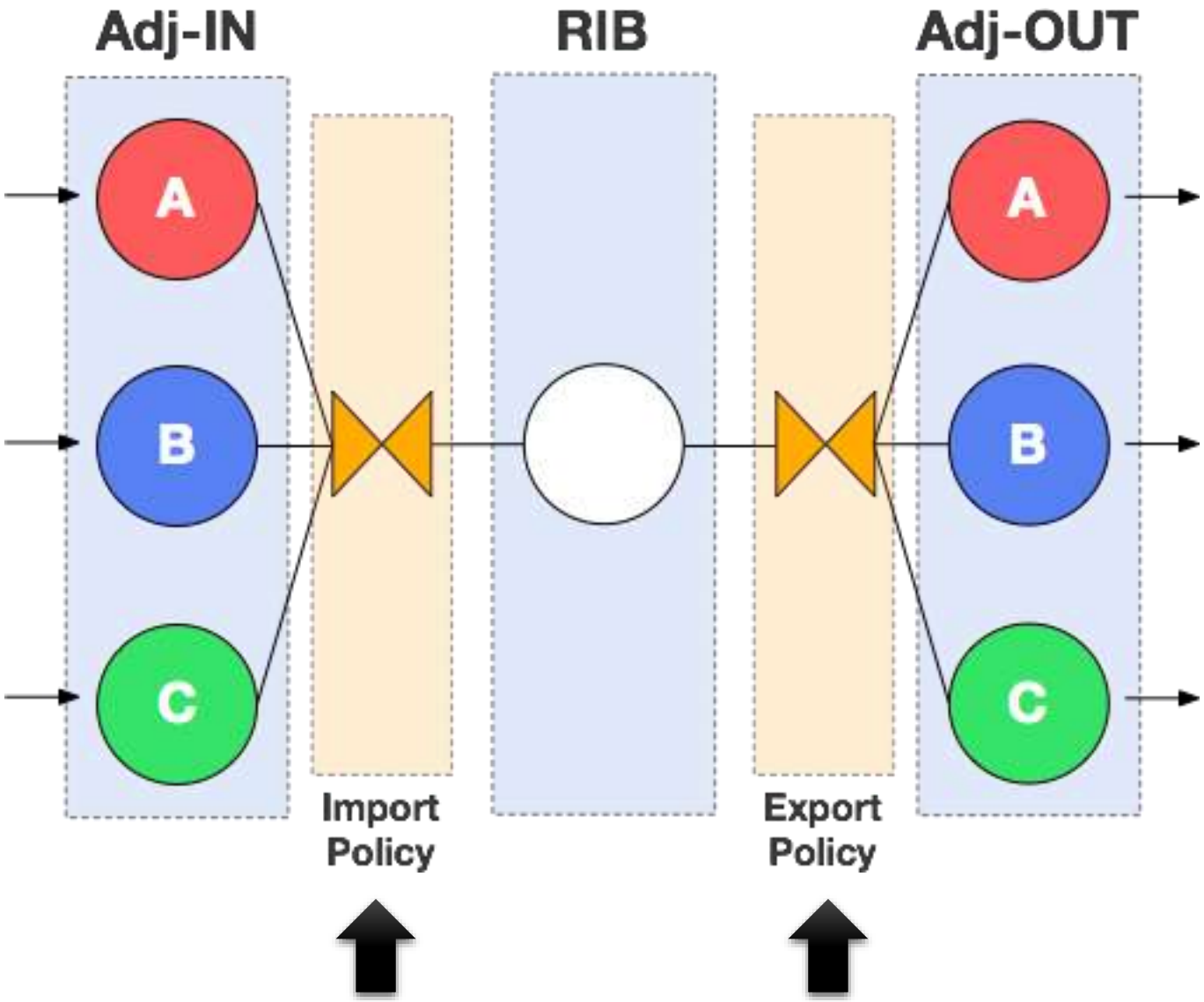
Policy definition structure



Policy configuration basics

- First you define policies
  - and defined-sets if needed to achieve your needs
- Then attach it to global config level
  - not in neighbor level
  - beware! There is a nob under neighbor level which is only for Route Server setup
- You can specify neighbor using `neighbor-set`
  - in import policy: from which neighbor
  - in export policy: to which neighbor

Where policies are applied



Other policy examples

- see the official doc for more details
  - https://github.com/osrg/gobgp/blob/master/docs/sources/policy.md

Install routes into Linux FIB / zebra integration

- gobgpd can act as a replacement of bgpd in quagga
- Setup zebra integration
  - And first we see BGP received route from **r3** installed in the routing table

Install routes into Linux FIB / zebra integration

```
zebra:
  config:
    enabled: true
    url: "unix:/var/run/quagga/zserv.api"
    redistribute-route-type-list: []
```

- Specify the path to `zserv.api` created by zebra daemon
  - gobgpd process needs rw access to it
    - Check permission if you encounter any problem
- `redistribute-route-type-list` specifies from which protocol gobgpd imports routes into BGP RIB
  - At this time we do not redistribute any route from zebra so just leave it blank

Install routes into Linux FIB / zebra integration

```
### apply configuration change
g2 $ systemctl restart gobgpd
```

- You need to restart gobgpd
  - some configuration change need restart of gobgpd to take effect
  - zebra integration is one of them
- Booting order is important!
  - zebra process must be started before gobgpd starts

Install routes into Linux FIB / zebra integration

```
g2 $ vtysh -d zebra -c "show ip route"
Codes: K - kernel route, C - connected, S - static, R - RIP,
       O - OSPF, I - IS-IS, B - BGP, A - Babel,
       > - selected route, * - FIB route

K>* 0.0.0.0/0 via 10.0.2.2, eth0
S>* 10.0.0.0/8 [1/0] is directly connected, Null0, bh
C>* 10.0.2.0/24 is directly connected, eth0
S>* 10.1.0.0/16 [1/0] is directly connected, Null0, bh
O>* 10.1.0.101/32 [110/10] via 10.1.12.101, eth3, 1d02h44m
O   10.1.0.102/32 [110/10] is directly connected, lo, 1d02h44m
C>* 10.1.0.102/32 is directly connected, lo
O   10.1.12.0/24 [110/10] is directly connected, eth3, 1d02h44m
C>* 10.1.12.0/24 is directly connected, eth3
O>* 10.1.14.0/24 [110/0] via 10.1.12.101, eth3, 1d02h44m
B>* 10.3.0.0/16 [20/0] via 10.173.176.103, eth2, 00:16:10
C>* 10.173.176.0/24 is directly connected, eth2
C>* 10.254.0.0/24 is directly connected, eth1
C>* 127.0.0.0/8 is directly connected, lo
S>* 172.16.0.0/12 [1/0] is directly connected, Null0, bh
S>* 192.168.0.0/16 [1/0] is directly connected, Null0, bh
```

Originate routes from zebra

- Inject routes from zebra into gobgpd then advertise to **r3**
  - We use static null route
    - Common way to originate your own prefixes
- Set import policy to select only routes that we want to import

Originate routes from zebra

```
zebra:
  config:
    enabled: true
    url: "unix:/var/run/quagga/zserv.api"
    redistribute-route-type-list:
      - static
```

- `redistribute-route-type-list`
  - Specify from which protocol gobgpd imports routes
- If you like to also redistribute connected routes and ospf routes then just append them to the list

Originate routes from zebra

```
### apply configuration change, need restart
g2 $ systemctl restart gobgpd

### all static routes are imported
g2 $ gobgp global rib
    Network              Next Hop             AS_PATH              Age
Attrs
*>  10.0.0.0/8          0.0.0.0
00:00:05   [{Origin: i} {Med: 0}]
*>  10.1.0.0/16         0.0.0.0
00:00:05   [{Origin: i} {Med: 0}]
*>  172.16.0.0/12       0.0.0.0
00:00:05   [{Origin: i} {Med: 0}]
*>  192.168.0.0/16      0.0.0.0
00:00:05   [{Origin: i} {Med: 0}]

g2 $ vtysh -d zebra -c "show ip route static“
S>* 10.0.0.0/8 [1/0] is directly connected, Null0, bh
S>* 10.1.0.0/16 [1/0] is directly connected, Null0, bh
S>* 172.16.0.0/12 [1/0] is directly connected, Null0, bh
S>* 192.168.0.0/16 [1/0] is directly connected, Null0, bh
```

Originate routes from zebra

```
defined-sets:
  prefix-sets:
    - prefix-set-name: my-prefixes
      prefix-list:
        - ip-prefix: 10.1.0.0/16

policy-definitions:
  - name: zebra-import-my-prefixes
    statements:
      - conditions:
          bgp-conditions:
            route-type: local
          match-prefix-set:
            prefix-set: my-prefixes
            match-set-options: invert
        actions:
          route-disposition: reject-route
```

- Reject any routes from zebra except my-prefixes
  - `route-type: local` matches routes from zebra

Originate routes from zebra

```
global:
  config:
    as: 65001
    router-id: 10.1.0.102
  apply-policy:
    config:
      import-policy-list:
        - zebra-import-my-prefixes
        - tag-ixp-neighbors
      default-import-policy: accept-route
```

```
### apply configuration change
g2 $ systemctl restart gobgpd
```

- This seems to be just a policy change but you still need to restart gobgpd to get routes injected from zebra installed into BGP RIB
  - currently a kind of `soft reset in` from zebra is not supported!
  - If you really don't like to restart gobgpd, then delete/re-add those static routes in zebra console
    - and open an issue at GitHub ☺

Originate routes from zebra

```
g2 $ gobgp global rib
    Network              Next Hop             AS_PATH              Age
Attrs
*>  10.1.0.0/16         0.0.0.0
00:37:38   [{Origin: i} {Med: 0}]
*>  10.3.0.0/16         10.173.176.103       65003
00:37:20   [{Origin: i} {Communities: 65001:1000}]
```

```
root@r3> show route protocol bgp

inet.0: 11 destinations, 13 routes (11 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

10.1.0.0/16        *[BGP/170] 00:39:15, MED 0, localpref 100
                      AS path: 65001 I
                    > to 10.173.176.102 via ge-0/0/1.0
```

- We can see my route 10.1.0.0/16 appears in BGP RIB and advertised to **r3**

Establish iBGP

- Establish iBGP peer between **g2** and **r1** using local loopback addresses
- Apply export policy
  - to do `next hop self`
  - to all routes received from **r3** which is peer on IXP

Establish iBGP

```
neighbors:
  - config:
      neighbor-address: 10.1.0.101
      peer-as: 65001
    transport:
      config:
        local-address: 10.1.0.102
```

```
### apply configuration change
g2 $ systemctl reload gobgpd

### check establishment
g2 $ gobgp neigh
Peer              AS  Up/Down State       |#Received  Accepted
10.1.0.101     65001 00:07:41 Establ      |        1         1
10.173.176.103 65003 00:08:16 Establ      |        1         1
```

Establish iBGP

```
### check RIB
g2 $ gobgp global rib
    Network              Next Hop             AS_PATH              Age
Attrs
*>  10.1.0.0/16         0.0.0.0
00:09:02   [{Origin: i} {Med: 0}]
*>  10.3.0.0/16         10.173.176.103       65003
00:08:47   [{Origin: i} {Communities: 65001:1000}]
*>  10.4.0.0/16         10.1.14.104          65004
00:08:12   [{Origin: i} {LocalPref: 100}]

### check routing table
g2 $ # vtysh -d zebra -c "show ip route"
 : (snip)

O>* 10.1.14.0/24 [110/0] via 10.1.12.101, eth3, 1d04h50m
B>* 10.3.0.0/16 [20/0] via 10.173.176.103, eth2, 00:15:05
B>  10.4.0.0/16 [200/0] via 10.1.14.104 (recursive), 00:14:30
  *                       via 10.1.12.101, eth3, 00:14:30
 : (snip)
```

Establish iBGP

```
### check adj-out
g2 $ gobgp neighbor 10.1.0.101 adj-out
    Network              Next Hop             AS_PATH              Attrs
    10.1.0.0/16          10.1.0.102
[{Origin: i} {Med: 0} {LocalPref: 100}]
    10.3.0.0/16          10.173.176.103       65003
[{Origin: i} {LocalPref: 100} {Communities: 65001:1000}]
```

```
root@r1> show route protocol bgp
 : (snip)
10.3.0.0/16        *[BGP/170] 00:21:05, localpref 100, from 10.1.0.102
                      AS path: 65003 I
                    Discard
 : (snip)
```

- We'd like to change next-hop…
  - because IXP subnet address 10.172.176.0/24 is not in IGP (in this example it is OSPF)
  - **r1** cannot reach

Establish iBGP

```
defined-sets:
  bgp-defined-sets:
    community-sets:
      - community-set-name: from-ixp-neighbors
        community-list:
          - "65001:1000"

policy-definitions:
  - name: set-next-hop-self
    statements:
      - conditions:
          bgp-conditions:
            match-community-set:
              community-set: from-ixp-neighbors
        actions:
          bgp-actions:
            set-next-hop: self
```

- Not possible to specify “from who” in export policy
- instead we can use community which we applied previously

Establish iBGP

```
global:
  apply-policy:
    config:
      export-policy-list:
        - set-next-hop-self
      default-export-policy: accept-route
```

```
### apply configuration change
g2 $ systemctl reload gobgpd

### perform soft reset out
g2 $ gobgp neighbor 10.1.0.101 softresetout
```

- When modifying export policy, you need to call `softresetout` manually
  - contrary to `softresetin` which will be performed automatically
- Caveats: currently after `softresetout` gobgpd will send ALL NLRIs in the RIB to peers, not only updated NLRIs
  - Maybe problematic with huge number of prefixes
  - Open an issue at GitHub :P

Establish iBGP

```
g2 $ gobgp neighbor 10.1.0.101 adj-out
    Network              Next Hop             AS_PATH              Attrs
    10.1.0.0/16          10.1.0.102
[{Origin: i} {Med: 0} {LocalPref: 100}]
    10.3.0.0/16          10.1.0.102           65003
[{Origin: i} {LocalPref: 100} {Communities: 65001:1000}]
```

```
root@r1> show route receive-protocol bgp 10.1.0.102

inet.0: 16 destinations, 18 routes (16 active, 0 holddown, 0 hidden)
  Prefix                  Nexthop              MED     Lclpref    AS
path
* 10.1.0.0/16            10.1.0.102           0       100        I
* 10.3.0.0/16            10.1.0.102                   100        65003
I
```

- next-hop attribute is modified as intended to point the loopback address of **g2**

Establish iBGP

```
root@r4> ping 10.3.0.103 source 10.4.0.104 count 3
PING 10.3.0.103 (10.3.0.103): 56 data bytes
64 bytes from 10.3.0.103: icmp_seq=0 ttl=62 time=10.496 ms
64 bytes from 10.3.0.103: icmp_seq=1 ttl=62 time=2.417 ms
64 bytes from 10.3.0.103: icmp_seq=2 ttl=62 time=2.586 ms

--- 10.3.0.103 ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max/stddev = 2.417/5.166/10.496/3.769 ms

root@r3> ping 10.4.0.104 source 10.3.0.103 count 3
PING 10.4.0.104 (10.4.0.104): 56 data bytes
64 bytes from 10.4.0.104: icmp_seq=0 ttl=62 time=10.624 ms
64 bytes from 10.4.0.104: icmp_seq=1 ttl=62 time=1.948 ms
 : (snip)
```

- Now you can reach from **r4** loopback address to **r3** loopback address that all RIBs/FIBs in between them are properly set up

Add peer to Route Server

- Establish eBGP peer between **g2** and **rs**, which is Route Server at IXP
- Control advertisement policy
  - by adding specific communities
  - Here we add community value which will reject advertising our prefix only to **r3** from route server

Add peer to Route Server

```
neighbors:
  - config:
      neighbor-address: 10.173.176.211
      peer-as: 64686
      auth-password: pass65001
```

```
### apply configuration change
g2 $ systemctl reload gobgpd

### check establishment
g2 $ gobgp n
Peer              AS  Up/Down State       |#Received  Accepted
10.1.0.101     65001 03:16:11 Establ      |        1         1
10.173.176.103 65003 03:16:46 Establ      |        1         1
10.173.176.211 64686 00:05:32 Establ      |        1         1
```

- Nothing new here except md5 password authentication

Add peer to Route Server

```
g2 $ gobgp global rib
    Network              Next Hop             AS_PATH              Age
Attrs
*>  10.1.0.0/16         0.0.0.0
03:21:07   [{Origin: i} {Med: 0}]
*>  10.3.0.0/16         10.173.176.103       65003
03:20:52   [{Origin: i} {Communities: 65001:1000}]
*   10.3.0.0/16         10.173.176.103       65003
00:09:38   [{Origin: i}]
*>  10.4.0.0/16         10.1.14.104          65004
03:20:17   [{Origin: i} {LocalPref: 100}]
```

- Receiving the same route for AS65003 from rs
  - It's actually coming from **rs** that AS_PATH does not include 64686 which is the ASN of **rs**
- No communities for IXP peers 65001:1000 added yet
  - We'll do that later

Add peer to Route Server

```
root@r3> show route receive-protocol bgp 10.173.176.211

inet.0: 12 destinations, 16 routes (12 active, 0 holddown, 0 hidden)
  Prefix                  Nexthop              MED     Lclpref    AS
path
  10.1.0.0/16             10.173.176.102       0                  65001
I
  10.4.0.0/16             10.173.176.102                          65001
65004 I
```

- Receiving routes from **rs** at **r3**
  - AS_PATH does not contain 64686
  - next-hop is not the address of **rs** which is 10.173.176.211

Add peer to Route Server

```
defined-sets:
  neighbor-sets:
    - neighbor-set-name: ixp-neighbors
      neighbor-info-list:
        - 10.173.176.103
        - 10.173.176.211
```

```
### apply configuration
g2 $ systemctl reload gobgpd
```

- Add the same community to tag routes `coming from IXP peers` as well as **r3**
- We've already have policy for IXP peers applied to **r3**
  - What we need to do here is just add the neighbor address of the **rs** to the neighbor-set

Add peer to Route Server

```
### apply configuration change
g2 $ systemctl reload gobgpd
g2 $ gobgp global rib
    Network              Next Hop             AS_PATH              Age
Attrs
*>  10.1.0.0/16         0.0.0.0
03:51:52   [{Origin: i} {Med: 0}]
*>  10.3.0.0/16         10.173.176.103       65003
03:51:37   [{Origin: i} {Communities: 65001:1000}]
*   10.3.0.0/16         10.173.176.103       65003
00:40:23   [{Origin: i} {Communities: 65001:1000}]
*>  10.4.0.0/16         10.1.14.104          65004
03:51:02   [{Origin: i} {LocalPref: 100}]
```

- Routes received from **rs** are also tagged with community 65001:1000

Control Route Server policy with communities

- IXP in this tutorial provides following policy at Route Server to control advertisement to other peers

ASN	meaning
0:N	Do not advertise to N
64686:N	Advertise to N
0:64686	Do not advertise to ANY peer

- ex.) If you like to advertise to all peers EXCEPT AS65003
  - Add “0:65003”
- ex.) If you like to advertise ONLY TO AS65003 and AS65123
  - Add “0:64686 64686:65003 64686:65123”

Control Route Server policy with communities

```
defined-sets:
  neighbor-sets:
    - neighbor-set-name: ixp-rs
      neighbor-info-list:
        - 10.173.176.211

policy-definitions:
  - name: rs-no-export-to-as65003
    statements:
      - conditions:
          match-neighbor-set:
            neighbor-set: ixp-rs
        actions:
          bgp-actions:
            set-community:
              options: add
              set-community-method:
                communities-list:
                  - "0:65003"
```

Control Route Server policy with communities

```
global:
  config:
    as: 65001
    router-id: 10.1.0.102
  apply-policy:
    config:
      import-policy-list:
        - zebra-import-my-prefixes
        - tag-ixp-neighbors
      default-import-policy: accept-route
      export-policy-list:
        - set-next-hop-self
        - rs-no-export-to-as65003
      default-export-policy: accept-route
```

```
g2 $ systemctl reload gobgpd
g2 $ systemctl neighbor 10.173.176.211 softresetout
```

- Remember that you need to call `softresetout` manually when you change export policy

Control Route Server policy with communities

```
g2 $ gobgp neighbor 10.173.176.211 adj-out
    Network              Next Hop             AS_PATH              Attrs
    10.1.0.0/16          10.173.176.102       65001
[{Origin: i} {Med: 0} {Communities: 0:65003}]
    10.3.0.0/16          10.173.176.102       65001 65003
[{Origin: i} {Communities: 65001:1000, 0:65003}]
    10.4.0.0/16          10.173.176.102       65001 65004
[{Origin: i} {Communities: 0:65003}]
```

```
root@r3> show route receive-protocol bgp 10.173.176.211

inet.0: 12 destinations, 14 routes (12 active, 0 holddown, 0 hidden)
```

- We can confirm that r3 does not receive any routes from **rs** because of the community based policy control has taken place

RPKI

- Configure route validation with RPKI
- Apply policy based on validation result
  - tag with communities
  - We will use following value

RPKI validation result	community to add
Valid	65001:2001
Invalid	65001:2002
Not found	65001:2003

RPKI

```
rpki-servers:
  - config:
      address: 210.173.170.254
      port: 323
```

```
g2 $ systemctl restart gobgpd
```

- What we need is only the address of ROA cache server
- Here we use open ROA cache server operated by INTERNET MULTIFEED CO.
  - see http://www.mfeed.ad.jp/rpki/en/
- This change requires restart of gobgpd

RPKI

```
g2 $ gobgp rpki server
Session                 State  Uptime     #IPv4/IPv6 records
210.173.170.254:323     Up     00:05:38   24977/3522

g2 $ gobgp global rib
    Network             Next Hop             AS_PATH              Age
Attrs
*>  10.1.0.0/16         0.0.0.0
00:05:48   [{Origin: i} {Med: 0}]
N*> 10.3.0.0/16         10.173.176.103       65003
00:05:38   [{Origin: i} {Communities: 65001:1000}]
N*  10.3.0.0/16         10.173.176.103       65003
00:05:35   [{Origin: i} {Communities: 65001:1000}]
N*> 10.4.0.0/16         10.1.14.104          65004
00:05:30   [{Origin: i} {LocalPref: 100}]
```

- “N” indicates record not found in ROA table
  - Of course it’s a private IP!
- Local originating route will not be validated, only received routes are

RPKI

```
policy-definitions:
  - name: tag-rpki-validation
    statements:
      - conditions:
          bgp-conditions:
            rpki-validation-result: valid
        actions:
          bgp-actions:
            set-community:
              options: add
              set-community-method:
                communities-list:
                  - "65001:2001“

# ...
```

- You can use `rpki—validation-result` to match against RPKI validation status of the route

RPKI

```
# cont'd
      - conditions:
          bgp-conditions:
            rpki-validation-result: invalid
        actions:
          bgp-actions:
            set-community:
              options: add
              set-community-method:
                communities-list:
                  - "65001:2002"
      - conditions:
          bgp-conditions:
            rpki-validation-result: not-found
        actions:
          bgp-actions:
            set-community:
              options: add
              set-community-method:
                communities-list:
                  - "65001:2003"
```

RPKI

```
global:
  apply-policy:
    config:
      import-policy-list:
        - zebra-import-my-prefixes
        - tag-ixp-neighbors
        - tag-rpki-validation
```

```
g2 $ systemctl restart gobgpd
g2 $ gobgp global rib
    Network              Next Hop             AS_PATH              Age
Attrs
*>  10.1.0.0/16         0.0.0.0
02:20:52   [{Origin: i} {Med: 0}]
N*> 10.3.0.0/16         10.173.176.103       65003
02:20:42   [{Origin: i} {Communities: 65001:2003, 65001:1000}]
N*  10.3.0.0/16         10.173.176.103       65003
02:20:39   [{Origin: i} {Communities: 65001:2003, 65001:1000}]
N*> 10.4.0.0/16         10.1.14.104          65004
02:20:34   [{Origin: i} {LocalPref: 100} {Communities: 65001:2003}]
```

MRT

- Configure MRT dump to monitor
  - RIB periodically (TABLE_DUMPv2)
  - incoming UPDATE messages (BGP4MP)

MRT

```
mrt-dump:
  - config:
      dump-type: updates
      file-name: /tmp/updates.2006-01-02_1504.mrt
      rotation-interval: 180
  - config:
      dump-type: table
      file-name: /tmp/table.mrt
      dump-interval: 60
```

```
g2 $ systemctl restart gobgpd
```

- Dump of update messages can be rotated automatically by specifying interval and file name format
  - 2006-01-02_1504 = YYYY-MM-DD_HHMM
  - Seems weird but this is how to specify dates format
- Table dump currently does not support rotating
  - content will be replaced every `dump-interval` second

MRT

```
g2 $ ls -al /tmp/*.mrt
-rw-r--r-- 1 quagga quagga 1768 Dec  3 11:36 /tmp/table.mrt
-rw-r--r-- 1 quagga quagga  241 Dec  3 11:28 /tmp/updates.2016-12-
03_1128.mrt
-rw-r--r-- 1 quagga quagga    0 Dec  3 11:31 /tmp/updates.2016-12-
03_1131.mrt
-rw-r--r-- 1 quagga quagga    0 Dec  3 11:34 /tmp/updates.2016-12-
03_1134.mrt
```

- Dumps are created
- Use your favorite tool to see them
  - `bgpdump` from bgptools
  - `bgpreader` from BGPStream
  - etc…

BMP

- BMP is a protocol to monitor BGP
  - Abbreviation of BGP Monitoring Protocol
  - RFC7854
- GoBGP can export BMP messages
- Several implementation for BMP server can be found
  - GoBGP itself can be simple BMP server
  - Here we use Ryu BMP Server
    - http://osrg.github.io/bmp/

BMP

```
bmp-servers:
  - config:
      address: 10.254.0.250
      port: 11019
```

```
g2 $ systemctl restart gobgpd
```

```
util $ docker run -it -p 11019:11019 osrg/ryu /bin/bash
```

```
util(container) # ryu run --verbose ./ryu/ryu/app/bmpstation
loading app ./ryu/ryu/app/bmpstation
instantiating app ./ryu/ryu/app/bmpstation of BMPStation
BRICK bmpstation
listening on 0.0.0.0:11019
```

- What we need is only the address of BMP Server
  - We will run Ryu BMP Server on **util**
- Here again it requires the restart of gobgpd
- Launch Ryu BMP Server within docker container

BMP

```
g2 $ gobgp neigh 10.173.176.103 reset
```

```
### (cont'd util(container) console)
2016 Dec 03 13:43:29 | 10.254.0.102 | BMPInitiation(info=[],len=6,type=4,version=3)

2016 Dec 03 13:43:29 | 10.254.0.102 |
BMPPeerUpNotification(is_post_policy=False,len=158,local_address='0.0.0.0',local_port=49071,pee
r_address='0.0.0.0',peer_as=64686,peer_bgp_id='10.173.176.211',peer_distinguisher=0,peer_type=0
,received_open_message=BGPOpen(bgp_identifier='10.173.176.211',hold_time=90,len=45,my_as=64686,
opt_param=[BGPOptParamCapabilityRouteRefresh(cap_code=2,cap_length=0,length=2,type=2),
BGPOptParamCapabilityMultiprotocol(afi=1,cap_code=1,cap_length=4,length=6,reserved=0,safi=1,typ
e=2),
BGPOptParamCapabilityFourOctetAsNumber(as_number=64686,cap_code=65,cap_length=4,length=6,type=2
)],opt_param_len=16,type=1,version=4),remote_port=179,sent_open_message=BGPOpen(bgp_identifier=
'10.1.0.102',hold_time=90,len=45,my_as=65001,opt_param=[BGPOptParamCapabilityRouteRefresh(cap_c
ode=2,cap_length=0,length=2,type=2),
BGPOptParamCapabilityMultiprotocol(afi=1,cap_code=1,cap_length=4,length=6,reserved=0,safi=1,typ
e=2),
BGPOptParamCapabilityFourOctetAsNumber(as_number=65001,cap_code=65,cap_length=4,length=6,type=2
)],opt_param_len=16,type=1,version=4),timestamp=1480772609.0,type=3,version=3)

 : (snip)
```

- We can see BMP message received at **util**

Store BGP event log into InfluxDB

- InfluxDB is one of the time series DB
  - like RRD
- GoBGP can export BGP related event logs directly into InfluxDB
- This function is not documented yet

Store BGP event log into InfluxDB

```
### prepare influxdb on util
util $ docker run --name=influxdb -d -p 8086:8086 influxdb
util $ docker run --rm --net=container:influxdb -it influxdb influx -
host localhost
Visit https://enterprise.influxdata.com to register for updates,
InfluxDB server management, and monitoring.
Connected to http://localhost:8086 version 1.1.0
InfluxDB shell version: 1.1.0
> create database gobgpd
> show databases
name: databases
name
----
_internal
gobgpd
```

- Create new database

Store BGP event log into InfluxDB

```
collector:
  config:
    url: http://10.254.0.250:8086
    db-name: gobgpd
```

```
g2 $ systemctl restart gobgpd
```

- What we need is only the address of InfluxDB
- Here again it requires the restart of gobgpd
  - Note that you first need to launch InfluxDB because gobgpd will not boot if it fails to connect
- There is a bug!
  - If you enable this, import policy against zebra will not work properly

Store BGP event log into InfluxDB

```
### util(docker)/influx console (cont'd)
> use gobgpd
Using database gobgpd
> select * from peer
name: peer
time                    PeerAS  PeerAddress     PeerID          State
----                    ------  -----------     ------          -----
1480776860991000000     64686   10.173.176.211  10.173.176.211
Established
1480776861015000000     65001   10.1.0.101      10.1.0.101
Established
1480776867995000000     65003   10.173.176.103  10.3.0.103
Established
> select * from updates
 : (snip)
```

- You can see records inserted into InfluxDB

Wrap up

- You can find further information in official docs
  - https://github.com/osrg/gobgp/tree/master/docs/sources