Activates the following debug messages:

- Data plane received / send messages
- Connection events

debug bfd network
Toggle network events: show messages about socket failures and unexpected BFD messages that may not belong to registered peers.

debug bfd peer
Toggle peer event log messages: show messages about peer creation/removal and state changes.

debug bfd zebra
Toggle zebra message events: show messages about interfaces, local addresses, VRF and daemon peer registrations.

3.3 BGP

BGP stands for Border Gateway Protocol. The latest BGP version is 4. BGP-4 is one of the Exterior Gateway Protocols and the de facto standard interdomain routing protocol. BGP-4 is described in **RFC 1771** and updated by **RFC 4271**. **RFC 2858** adds multiprotocol support to BGP-4.

3.3.1 Starting BGP

The default configuration file of *bgpd* is `bgpd.conf`. *bgpd* searches the current directory first, followed by /etc/frr/bgpd.conf. All of *bgpd*'s commands must be configured in `bgpd.conf` when the integrated config is not being used.

bgpd specific invocation options are described below. Common options may also be specified (*Common Invocation Options*).

-p, --bgp_port <port>
Set the bgp protocol's port number. When port number is 0, that means do not listen bgp port.

-l, --listenon
Specify specific IP addresses for bgpd to listen on, rather than its default of `0.0.0.0` / `::`. This can be useful to constrain bgpd to an internal address, or to run multiple bgpd processes on one host. Multiple addresses can be specified.

In the following example, bgpd is started listening for connections on the addresses 100.0.1.2 and fd00::2:2. The options -d (runs in daemon mode) and -f (uses specific configuration file) are also used in this example as we are likely to run multiple bgpd instances, each one with different configurations, when using -l option.

Note that this option implies the –no_kernel option, and no learned routes will be installed into the linux kernel.

```
# /usr/lib/frr/bgpd -d -f /some-folder/bgpd.conf -l 100.0.1.2 -l fd00::2:2
```

-n, --no_kernel
Do not install learned routes into the linux kernel. This option is useful for a route-reflector environment or if you are running multiple bgp processes in the same namespace. This option is different than the –no_zebra option in that a ZAPI connection is made.

This option can also be toggled during runtime by using the `[no] bgp no-rib` commands in VTY shell.

Note that this option will persist after saving the configuration during runtime, unless unset by the `no bgp no-rib` command in VTY shell prior to a configuration write operation.

-S, --skip_runas
Skip the normal process of checking capabilities and changing user and group information.

-e, --ecmp
Run BGP with a limited ecmp capability, that is different than what BGP was compiled with. The value specified must be greater than 0 and less than or equal to the MULTIPATH_NUM specified on compilation.

-Z, --no_zebra
Do not communicate with zebra at all. This is different than the –no_kernel option in that we do not even open a ZAPI connection to the zebra process.

-s, --socket_size
When opening tcp connections to our peers, set the socket send buffer size that the kernel will use for the peers socket. This option is only really useful at a very large scale. Experimentation should be done to see if this is helping or not at the scale you are running at.

LABEL MANAGER

-I, --int_num
Set zclient id. This is required when using Zebra label manager in proxy mode.

3.3.2 Basic Concepts

Autonomous Systems

From **RFC 1930**:

> An AS is a connected group of one or more IP prefixes run by one or more network operators which has a SINGLE and CLEARLY DEFINED routing policy.

Each AS has an identifying number associated with it called an ASN (Autonomous System Number). This is a two octet value ranging in value from 1 to 65535. The AS numbers 64512 through 65535 are defined as private AS numbers. Private AS numbers must not be advertised on the global Internet.

The ASN is one of the essential elements of BGP. BGP is a distance vector routing protocol, and the AS-Path framework provides distance vector metric and loop detection to BGP.

See also:

RFC 1930

Address Families

Multiprotocol extensions enable BGP to carry routing information for multiple network layer protocols. BGP supports an Address Family Identifier (AFI) for IPv4 and IPv6. Support is also provided for multiple sets of per-AFI information via the BGP Subsequent Address Family Identifier (SAFI). FRR supports SAFIs for unicast information, labeled information (**RFC 3107** and **RFC 8277**), and Layer 3 VPN information (**RFC 4364** and **RFC 4659**).

Route Selection

The route selection process used by FRR's BGP implementation uses the following decision criterion, starting at the top of the list and going towards the bottom until one of the factors can be used.

1. **Weight check**

   Prefer higher local weight routes to lower routes.

2. **Local preference check**

   Prefer higher local preference routes to lower.

   If `bgp bestpath aigp` is enabled, and both paths that are compared have AIGP attribute, BGP uses AIGP tie-breaking unless both of the paths have the AIGP metric attribute. This means that the AIGP attribute is not evaluated during the best path selection process between two paths when one path does not have the AIGP attribute.

3. **Local route check**

   Prefer local routes (statics, aggregates, redistributed) to received routes.

4. **AS path length check**

   Prefer shortest hop-count AS_PATHs.

5. **Origin check**

   Prefer the lowest origin type route. That is, prefer IGP origin routes to EGP, to Incomplete routes.

6. **MED check**

   Where routes with a MED were received from the same AS, prefer the route with the lowest MED. *Multi-Exit Discriminator*.

7. **External check**

   Prefer the route received from an external, eBGP peer over routes received from other types of peers.

8. **IGP cost check**

   Prefer the route with the lower IGP cost.

9. **Multi-path check**

   If multi-pathing is enabled, then check whether the routes not yet distinguished in preference may be considered equal. If `bgp bestpath as-path multipath-relax` is set, all such routes are considered equal, otherwise routes received via iBGP with identical AS_PATHs or routes received from eBGP neighbours in the same AS are considered equal.

10. **Already-selected external check**

    Where both routes were received from eBGP peers, then prefer the route which is already selected. Note that this check is not applied if `bgp bestpath compare-routerid` is configured. This check can prevent some cases of oscillation.

11. **Router-ID check**

    Prefer the route with the lowest *router-ID*. If the route has an *ORIGINATOR_ID* attribute, through iBGP reflection, then that router ID is used, otherwise the *router-ID* of the peer the route was received from is used.

12. **Cluster-List length check**

    The route with the shortest cluster-list length is used. The cluster-list reflects the iBGP reflection path the route has taken.

13. **Peer address**

    Prefer the route received from the peer with the higher transport layer address, as a last-resort tie-breaker.

Capability Negotiation

When adding IPv6 routing information exchange feature to BGP. There were some proposals. IETF (Internet Engineering Task Force) IDR (Inter Domain Routing) adopted a proposal called Multiprotocol Extension for BGP. The specification is described in **RFC 2283**. The protocol does not define new protocols. It defines new attributes to existing BGP. When it is used exchanging IPv6 routing information it is called BGP-4+. When it is used for exchanging multicast routing information it is called MBGP.

bgpd supports Multiprotocol Extension for BGP. So if a remote peer supports the protocol, *bgpd* can exchange IPv6 and/or multicast routing information.

Traditional BGP did not have the feature to detect a remote peer's capabilities, e.g. whether it can handle prefix types other than IPv4 unicast routes. This was a big problem using Multiprotocol Extension for BGP in an operational network. **RFC 2842** adopted a feature called Capability Negotiation. *bgpd* use this Capability Negotiation to detect the remote peer's capabilities. If a peer is only configured as an IPv4 unicast neighbor, *bgpd* does not send these Capability Negotiation packets (at least not unless other optional BGP features require capability negotiation).

By default, FRR will bring up peering with minimal common capability for the both sides. For example, if the local router has unicast and multicast capabilities and the remote router only has unicast capability the local router will establish the connection with unicast only capability. When there are no common capabilities, FRR sends Unsupported Capability error and then resets the connection.

3.3.3 BGP Router Configuration

ASN and Router ID

First of all you must configure BGP router with the `router bgp ASN` command. The AS number is an identifier for the autonomous system. The AS identifier can either be a number or two numbers separated by a period. The BGP protocol uses the AS identifier for detecting whether the BGP connection is internal or external.

`router bgp ASN`

Enable a BGP protocol process with the specified ASN. After this statement you can input any *BGP Commands*.

`bgp router-id A.B.C.D`

This command specifies the router-ID. If *bgpd* connects to *zebra* it gets interface and address information. In that case default router ID value is selected as the largest IP Address of the interfaces. When *router zebra* is not enabled *bgpd* can't get interface information so *router-id* is set to 0.0.0.0. So please set router-id by hand.

Multiple Autonomous Systems

FRR's BGP implementation is capable of running multiple autonomous systems at once. Each configured AS corresponds to a *Virtual Routing and Forwarding*. In the past, to get the same functionality the network administrator had to run a new *bgpd* process; using VRFs allows multiple autonomous systems to be handled in a single process.

When using multiple autonomous systems, all router config blocks after the first one must specify a VRF to be the target of BGP's route selection. This VRF must be unique within respect to all other VRFs being used for the same purpose, i.e. two different autonomous systems cannot use the same VRF. However, the same AS can be used with different VRFs.

Note: The separated nature of VRFs makes it possible to peer a single *bgpd* process to itself, on one machine. Note that this can be done fully within BGP without a corresponding VRF in the kernel or Zebra, which enables some practical use cases such as *route reflectors* and route servers.

Configuration of additional autonomous systems, or of a router that targets a specific VRF, is accomplished with the following command:

`router bgp ASN vrf VRFNAME`

`VRFNAME` is matched against VRFs configured in the kernel. When `vrf VRFNAME` is not specified, the BGP protocol process belongs to the default VRF.

An example configuration with multiple autonomous systems might look like this:

```
router bgp 1
 neighbor 10.0.0.1 remote-as 20
 neighbor 10.0.0.2 remote-as 30
!
router bgp 2 vrf blue
 neighbor 10.0.0.3 remote-as 40
 neighbor 10.0.0.4 remote-as 50
!
router bgp 3 vrf red
 neighbor 10.0.0.5 remote-as 60
 neighbor 10.0.0.6 remote-as 70
...
```

See also:

VRF Route Leaking

See also:

Virtual Routing and Forwarding

Views

In addition to supporting multiple autonomous systems, FRR's BGP implementation also supports *views*.

BGP views are almost the same as normal BGP processes, except that routes selected by BGP are not installed into the kernel routing table. Each BGP view provides an independent set of routing information which is only distributed via BGP. Multiple views can be supported, and BGP view information is always independent from other routing protocols and Zebra/kernel routes. BGP views use the core instance (i.e., default VRF) for communication with peers.

`router bgp AS-NUMBER view NAME`

Make a new BGP view. You can use an arbitrary word for the `NAME`. Routes selected by the view are not installed into the kernel routing table.

With this command, you can setup Route Server like below.

```
!
router bgp 1 view 1
 neighbor 10.0.0.1 remote-as 2
 neighbor 10.0.0.2 remote-as 3
!
router bgp 2 view 2
```

(continues on next page)

(continued from previous page)

```
neighbor 10.0.0.3 remote-as 4
neighbor 10.0.0.4 remote-as 5
```

show [ip] bgp view NAME
Display the routing table of BGP view NAME.

Route Selection

bgp bestpath as-path confed
This command specifies that the length of confederation path sets and sequences should should be taken into account during the BGP best path decision process.

bgp bestpath as-path multipath-relax
This command specifies that BGP decision process should consider paths of equal AS_PATH length candidates for multipath computation. Without the knob, the entire AS_PATH must match for multipath computation.

bgp bestpath compare-routerid
Ensure that when comparing routes where both are equal on most metrics, including local-pref, AS_PATH length, IGP cost, MED, that the tie is broken based on router-ID.

If this option is enabled, then the already-selected check, where already selected eBGP routes are preferred, is skipped.

If a route has an *ORIGINATOR_ID* attribute because it has been reflected, that *ORIGINATOR_ID* will be used. Otherwise, the router-ID of the peer the route was received from will be used.

The advantage of this is that the route-selection (at this point) will be more deterministic. The disadvantage is that a few or even one lowest-ID router may attract all traffic to otherwise-equal paths because of this check. It may increase the possibility of MED or IGP oscillation, unless other measures were taken to avoid these. The exact behaviour will be sensitive to the iBGP and reflection topology.

bgp bestpath peer-type multipath-relax
This command specifies that BGP decision process should consider paths from all peers for multipath computation. If this option is enabled, paths learned from any of eBGP, iBGP, or confederation neighbors will be multipath if they are otherwise considered equal cost.

bgp bestpath aigp
Use the bgp bestpath aigp command to evaluate the AIGP attribute during the best path selection process between two paths that have the AIGP attribute.

When bgp bestpath aigp is disabled, BGP does not use AIGP tie-breaking rules unless paths have the AIGP attribute.

Disabled by default.

maximum-paths (1-128)
Sets the maximum-paths value used for ecmp calculations for this bgp instance in EBGP. The maximum value listed, 128, can be limited by the ecmp cli for bgp or if the daemon was compiled with a lower ecmp value. This value can also be set in ipv4/ipv6 unicast/labeled unicast to only affect those particular afi/safi's.

maximum-paths ibgp (1-128) [equal-cluster-length]
Sets the maximum-paths value used for ecmp calculations for this bgp instance in IBGP. The maximum value listed, 128, can be limited by the ecmp cli for bgp or if the daemon was compiled with a lower ecmp value. This value can also be set in ipv4/ipv6 unicast/labeled unicast to only affect those particular afi/safi's.

Administrative Distance Metrics

distance bgp (1-255) (1-255) (1-255)

This command changes distance value of BGP. The arguments are the distance values for external routes, internal routes and local routes respectively.

distance (1-255) A.B.C.D/M

distance (1-255) A.B.C.D/M WORD

Sets the administrative distance for a particular route.

Require policy on EBGP

bgp ebgp-requires-policy

This command requires incoming and outgoing filters to be applied for eBGP sessions as part of RFC-8212 compliance. Without the incoming filter, no routes will be accepted. Without the outgoing filter, no routes will be announced.

This is enabled by default for the traditional configuration and turned off by default for datacenter configuration.

When you enable/disable this option you MUST clear the session.

When the incoming or outgoing filter is missing you will see "(Policy)" sign under `show bgp summary`:

```
exit1# show bgp summary

IPv4 Unicast Summary (VRF default):
BGP router identifier 10.10.10.1, local AS number 65001 vrf-id 0
BGP table version 4
RIB entries 7, using 1344 bytes of memory
Peers 2, using 43 KiB of memory

Neighbor        V         AS   MsgRcvd   MsgSent   TblVer  InQ OutQ  Up/Down State/
↪PfxRcd   PfxSnt Desc
192.168.0.2     4      65002         8        10        0    0    0 00:03:09       ␣
↪    5 (Policy) N/A
fe80:1::2222    4      65002         9        11        0    0    0 00:03:09    ␣
↪(Policy) (Policy) N/A
```

Additionally a *show bgp neighbor* command would indicate in the *For address family:* block that:

```
exit1# show bgp neighbor
...
For address family: IPv4 Unicast
 Update group 1, subgroup 1
 Packet Queue length 0
 Inbound soft reconfiguration allowed
 Community attribute sent to this neighbor(all)
 Inbound updates discarded due to missing policy
 Outbound updates discarded due to missing policy
 0 accepted prefixes
```

Reject routes with AS_SET or AS_CONFED_SET types

`bgp reject-as-sets`

This command enables rejection of incoming and outgoing routes having AS_SET or AS_CONFED_SET type.

Suppress duplicate updates

`bgp suppress-duplicates`

For example, BGP routers can generate multiple identical announcements with empty community attributes if stripped at egress. This is an undesired behavior. Suppress duplicate updates if the route actually not changed. Default: enabled.

Send Hard Reset CEASE Notification for Administrative Reset

`bgp hard-administrative-reset`

Send Hard Reset CEASE Notification for 'Administrative Reset' events.

When disabled, and Graceful Restart Notification capability is exchanged between the peers, Graceful Restart procedures apply, and routes will be retained.

Enabled by default.

Disable checking if nexthop is connected on EBGP sessions

`bgp disable-ebgp-connected-route-check`

This command is used to disable the connection verification process for EBGP peering sessions that are reachable by a single hop but are configured on a loopback interface or otherwise configured with a non-directly connected IP address.

Route Flap Dampening

`bgp dampening (1-45) (1-20000) (1-50000) (1-255)`

This command enables BGP route-flap dampening and specifies dampening parameters.

half-life Half-life time for the penalty

reuse-threshold Value to start reusing a route

suppress-threshold Value to start suppressing a route

max-suppress Maximum duration to suppress a stable route

The route-flap damping algorithm is compatible with **RFC 2439**. The use of this command is not recommended nowadays.

At the moment, route-flap dampening is not working per VRF and is working only for IPv4 unicast and multicast.

See also:

https://www.ripe.net/publications/docs/ripe-378

Multi-Exit Discriminator

The BGP MED (Multi-Exit Discriminator) attribute has properties which can cause subtle convergence problems in BGP. These properties and problems have proven to be hard to understand, at least historically, and may still not be widely understood. The following attempts to collect together and present what is known about MED, to help operators and FRR users in designing and configuring their networks.

The BGP MED attribute is intended to allow one AS to indicate its preferences for its ingress points to another AS. The MED attribute will not be propagated on to another AS by the receiving AS - it is 'non-transitive' in the BGP sense.

E.g., if AS X and AS Y have 2 different BGP peering points, then AS X might set a MED of 100 on routes advertised at one and a MED of 200 at the other. When AS Y selects between otherwise equal routes to or via AS X, AS Y should prefer to take the path via the lower MED peering of 100 with AS X. Setting the MED allows an AS to influence the routing taken to it within another, neighbouring AS.

In this use of MED it is not really meaningful to compare the MED value on routes where the next AS on the paths differs. E.g., if AS Y also had a route for some destination via AS Z in addition to the routes from AS X, and AS Z had also set a MED, it wouldn't make sense for AS Y to compare AS Z's MED values to those of AS X. The MED values have been set by different administrators, with different frames of reference.

The default behaviour of BGP therefore is to not compare MED values across routes received from different neighbouring ASes. In FRR this is done by comparing the neighbouring, left-most AS in the received AS_PATHs of the routes and only comparing MED if those are the same.

Unfortunately, this behaviour of MED, of sometimes being compared across routes and sometimes not, depending on the properties of those other routes, means MED can cause the order of preference over all the routes to be undefined. That is, given routes A, B, and C, if A is preferred to B, and B is preferred to C, then a well-defined order should mean the preference is transitive (in the sense of orders[1]) and that A would be preferred to C.

However, when MED is involved this need not be the case. With MED it is possible that C is actually preferred over A. So A is preferred to B, B is preferred to C, but C is preferred to A. This can be true even where BGP defines a deterministic 'most preferred' route out of the full set of A,B,C. With MED, for any given set of routes there may be a deterministically preferred route, but there need not be any way to arrange them into any order of preference. With unmodified MED, the order of preference of routes literally becomes undefined.

That MED can induce non-transitive preferences over routes can cause issues. Firstly, it may be perceived to cause routing table churn locally at speakers; secondly, and more seriously, it may cause routing instability in iBGP topologies, where sets of speakers continually oscillate between different paths.

The first issue arises from how speakers often implement routing decisions. Though BGP defines a selection process that will deterministically select the same route as best at any given speaker, even with MED, that process requires evaluating all routes together. For performance and ease of implementation reasons, many implementations evaluate route preferences in a pair-wise fashion instead. Given there is no well-defined order when MED is involved, the best route that will be chosen becomes subject to implementation details, such as the order the routes are stored in. That may be (locally) non-deterministic, e.g.: it may be the order the routes were received in.

This indeterminism may be considered undesirable, though it need not cause problems. It may mean additional routing churn is perceived, as sometimes more updates may be produced than at other times in reaction to some event .

This first issue can be fixed with a more deterministic route selection that ensures routes are ordered by the neighbouring AS during selection. `bgp deterministic-med`. This may reduce the number of updates as routes are received, and may in some cases reduce routing churn. Though, it could equally deterministically produce the largest possible set of updates in response to the most common sequence of received updates.

[1] For some set of objects to have an order, there *must* be some binary ordering relation that is defined for *every* combination of those objects, and that relation *must* be transitive. I.e.:, if the relation operator is <, and if a < b and b < c then that relation must carry over and it *must* be that a < c for the objects to have an order. The ordering relation may allow for equality, i.e. a < b and b < a may both be true and imply that a and b are equal in the order and not distinguished by it, in which case the set has a partial order. Otherwise, if there is an order, all the objects have a distinct place in the order and the set has a total order)

A deterministic order of evaluation tends to imply an additional overhead of sorting over any set of n routes to a destination. The implementation of deterministic MED in FRR scales significantly worse than most sorting algorithms at present, with the number of paths to a given destination. That number is often low enough to not cause any issues, but where there are many paths, the deterministic comparison may quickly become increasingly expensive in terms of CPU.

Deterministic local evaluation can *not* fix the second, more major, issue of MED however. Which is that the non-transitive preference of routes MED can cause may lead to routing instability or oscillation across multiple speakers in iBGP topologies. This can occur with full-mesh iBGP, but is particularly problematic in non-full-mesh iBGP topologies that further reduce the routing information known to each speaker. This has primarily been documented with iBGP *route-reflection* topologies. However, any route-hiding technologies potentially could also exacerbate oscillation with MED.

This second issue occurs where speakers each have only a subset of routes, and there are cycles in the preferences between different combinations of routes - as the undefined order of preference of MED allows - and the routes are distributed in a way that causes the BGP speakers to 'chase' those cycles. This can occur even if all speakers use a deterministic order of evaluation in route selection.

E.g., speaker 4 in AS A might receive a route from speaker 2 in AS X, and from speaker 3 in AS Y; while speaker 5 in AS A might receive that route from speaker 1 in AS Y. AS Y might set a MED of 200 at speaker 1, and 100 at speaker 3. I.e, using ASN:ID:MED to label the speakers:

```
.
          /---------------\\
X:2------|--A:4-------A:5--|-Y:1:200
            Y:3:100--|-/   |
          \\---------------/
```

Assuming all other metrics are equal (AS_PATH, ORIGIN, 0 IGP costs), then based on the RFC4271 decision process speaker 4 will choose X:2 over Y:3:100, based on the lower ID of 2. Speaker 4 advertises X:2 to speaker 5. Speaker 5 will continue to prefer Y:1:200 based on the ID, and advertise this to speaker 4. Speaker 4 will now have the full set of routes, and the Y:1:200 it receives from 5 will beat X:2, but when speaker 4 compares Y:1:200 to Y:3:100 the MED check now becomes active as the ASes match, and now Y:3:100 is preferred. Speaker 4 therefore now advertises Y:3:100 to 5, which will also agrees that Y:3:100 is preferred to Y:1:200, and so withdraws the latter route from 4. Speaker 4 now has only X:2 and Y:3:100, and X:2 beats Y:3:100, and so speaker 4 implicitly updates its route to speaker 5 to X:2. Speaker 5 sees that Y:1:200 beats X:2 based on the ID, and advertises Y:1:200 to speaker 4, and the cycle continues.

The root cause is the lack of a clear order of preference caused by how MED sometimes is and sometimes is not compared, leading to this cycle in the preferences between the routes:

```
.
 /---> X:2 ---beats---> Y:3:100 --\\
|                                   |
|                                   |
 \\---beats--- Y:1:200 <---beats---/
```

This particular type of oscillation in full-mesh iBGP topologies can be avoided by speakers preferring already selected, external routes rather than choosing to update to new a route based on a post-MED metric (e.g. router-ID), at the cost of a non-deterministic selection process. FRR implements this, as do many other implementations, so long as it is not overridden by setting `bgp bestpath compare-routerid`, and see also *Route Selection*.

However, more complex and insidious cycles of oscillation are possible with iBGP route-reflection, which are not so easily avoided. These have been documented in various places. See, e.g.:

- [bgp-route-osci-cond]
- [stable-flexible-ibgp]

- [ibgp-correctness]

for concrete examples and further references.

There is as of this writing *no* known way to use MED for its original purpose; *and* reduce routing information in iBGP topologies; *and* be sure to avoid the instability problems of MED due the non-transitive routing preferences it can induce; in general on arbitrary networks.

There may be iBGP topology specific ways to reduce the instability risks, even while using MED, e.g.: by constraining the reflection topology and by tuning IGP costs between route-reflector clusters, see **RFC 3345** for details. In the near future, the Add-Path extension to BGP may also solve MED oscillation while still allowing MED to be used as intended, by distributing "best-paths per neighbour AS". This would be at the cost of distributing at least as many routes to all speakers as a full-mesh iBGP would, if not more, while also imposing similar CPU overheads as the "Deterministic MED" feature at each Add-Path reflector.

More generally, the instability problems that MED can introduce on more complex, non-full-mesh, iBGP topologies may be avoided either by:

- Setting *`bgp always-compare-med`*, however this allows MED to be compared across values set by different neighbour ASes, which may not produce coherent desirable results, of itself.
- Effectively ignoring MED by setting MED to the same value (e.g.: 0) using `set metric METRIC` on all received routes, in combination with setting *`bgp always-compare-med`* on all speakers. This is the simplest and most performant way to avoid MED oscillation issues, where an AS is happy not to allow neighbours to inject this problematic metric.

As MED is evaluated after the AS_PATH length check, another possible use for MED is for intra-AS steering of routes with equal AS_PATH length, as an extension of the last case above. As MED is evaluated before IGP metric, this can allow cold-potato routing to be implemented to send traffic to preferred hand-offs with neighbours, rather than the closest hand-off according to the IGP metric.

Note that even if action is taken to address the MED non-transitivity issues, other oscillations may still be possible. E.g., on IGP cost if iBGP and IGP topologies are at cross-purposes with each other - see the Flavel and Roughan paper above for an example. Hence the guideline that the iBGP topology should follow the IGP topology.

`bgp deterministic-med`

Carry out route-selection in way that produces deterministic answers locally, even in the face of MED and the lack of a well-defined order of preference it can induce on routes. Without this option the preferred route with MED may be determined largely by the order that routes were received in.

Setting this option will have a performance cost that may be noticeable when there are many routes for each destination. Currently in FRR it is implemented in a way that scales poorly as the number of routes per destination increases.

The default is that this option is not set.

Note that there are other sources of indeterminism in the route selection process, specifically, the preference for older and already selected routes from eBGP peers, *Route Selection*.

`bgp always-compare-med`

Always compare the MED on routes, even when they were received from different neighbouring ASes. Setting this option makes the order of preference of routes more defined, and should eliminate MED induced oscillations.

If using this option, it may also be desirable to use `set metric METRIC` to set MED to 0 on routes received from external neighbours.

This option can be used, together with `set metric METRIC` to use MED as an intra-AS metric to steer equal-length AS_PATH routes to, e.g., desired exit points.

Graceful Restart

BGP graceful restart functionality as defined in RFC-4724 defines the mechanisms that allows BGP speaker to continue to forward data packets along known routes while the routing protocol information is being restored.

Usually, when BGP on a router restarts, all the BGP peers detect that the session went down and then came up. This "down/up" transition results in a "routing flap" and causes BGP route re-computation, generation of BGP routing updates, and unnecessary churn to the forwarding tables.

The following functionality is provided by graceful restart:

1. The feature allows the restarting router to indicate to the helping peer the routes it can preserve in its forwarding plane during control plane restart by sending graceful restart capability in the OPEN message sent during session establishment.
2. The feature allows helping router to advertise to all other peers the routes received from the restarting router which are preserved in the forwarding plane of the restarting router during control plane restart.

```
(R1)-----------------------------------------------------------------(R2)

1. BGP Graceful Restart Capability exchanged between R1 & R2.

<---------------------------------------------------------------------->

2. Kill BGP Process at R1.

----------------------------------------------------------------------->

3. R2 Detects the above BGP Restart & verifies BGP Restarting
  Capability of R1.

4. Start BGP Process at R1.

5. Re-establish the BGP session between R1 & R2.

<---------------------------------------------------------------------->

6. R2 Send initial route updates, followed by End-Of-Rib.

<-----------------------------------------------------------------------

7. R1 was waiting for End-Of-Rib from R2 & which has been received
  now.

8. R1 now runs BGP Best-Path algorithm. Send Initial BGP  Update,
  followed by End-Of Rib

<---------------------------------------------------------------------->
```

BGP-GR Preserve-Forwarding State

BGP OPEN message carrying optional capabilities for Graceful Restart has 8 bit "Flags for Address Family" for given AFI and SAFI. This field contains bit flags relating to routes that were advertised with the given AFI and SAFI.

```
0 1 2 3 4 5 6 7
+-+-+-+-+-+-+-+-+
|F|   Reserved  |
+-+-+-+-+-+-+-+-+
```

The most significant bit is defined as the Forwarding State (F) bit, which can be used to indicate whether the forwarding state for routes that were advertised with the given AFI and SAFI has indeed been preserved during the previous BGP restart. When set (value 1), the bit indicates that the forwarding state has been preserved. The remaining bits are reserved and MUST be set to zero by the sender and ignored by the receiver.

`bgp graceful-restart preserve-fw-state`

FRR gives us the option to enable/disable the "F" flag using this specific vty command. However, it doesn't have the option to enable/disable this flag only for specific AFI/SAFI i.e. when this command is used, it applied to all the supported AFI/SAFI combinations for this peer.

End-of-RIB (EOR) message

An UPDATE message with no reachable Network Layer Reachability Information (NLRI) and empty withdrawn NLRI is specified as the End-of-RIB marker that can be used by a BGP speaker to indicate to its peer the completion of the initial routing update after the session is established.

For the IPv4 unicast address family, the End-of-RIB marker is an UPDATE message with the minimum length. For any other address family, it is an UPDATE message that contains only the MP_UNREACH_NLRI attribute with no withdrawn routes for that <AFI, SAFI>.

Although the End-of-RIB marker is specified for the purpose of BGP graceful restart, it is noted that the generation of such a marker upon completion of the initial update would be useful for routing convergence in general, and thus the practice is recommended.

Route Selection Deferral Timer

Specifies the time the restarting router defers the route selection process after restart.

Restarting Router : The usage of route election deferral timer is specified in https://tools.ietf.org/html/rfc4724#section-4.1

Once the session between the Restarting Speaker and the Receiving Speaker is re-established, the Restarting Speaker will receive and process BGP messages from its peers.

However, it MUST defer route selection for an address family until it either.

1. Receives the End-of-RIB marker from all its peers (excluding the ones with the "Restart State" bit set in the received capability and excluding the ones that do not advertise the graceful restart capability).
2. The Selection_Deferral_Timer timeout.

`bgp graceful-restart select-defer-time (0-3600)`
This is command, will set deferral time to value specified.

`bgp graceful-restart rib-stale-time (1-3600)`
This is command, will set the time for which stale routes are kept in RIB.

`bgp graceful-restart restart-time (0-4095)`

Set the time to wait to delete stale routes before a BGP open message is received.

Using with Long-lived Graceful Restart capability, this is recommended setting this timer to 0 and control stale routes with `bgp long-lived-graceful-restart stale-time`.

Default value is 120.

`bgp graceful-restart stalepath-time (1-4095)`

This is command, will set the max time (in seconds) to hold onto restarting peer's stale paths.

It also controls Enhanced Route-Refresh timer.

If this command is configured and the router does not receive a Route-Refresh EoRR message, the router removes the stale routes from the BGP table after the timer expires. The stale path timer is started when the router receives a Route-Refresh BoRR message.

`bgp graceful-restart notification`

Indicate Graceful Restart support for BGP NOTIFICATION messages.

After changing this parameter, you have to reset the peers in order to advertise N-bit in Graceful Restart capability.

Without Graceful-Restart Notification capability (N-bit not set), GR is not activated when receiving CEASE/HOLDTIME expire notifications.

When sending `CEASE/Administrative Reset` (`clear bgp`), the session is closed and routes are not retained. When N-bit is set and `bgp hard-administrative-reset` is turned off Graceful-Restart is activated and routes are retained.

Enabled by default.

BGP Per Peer Graceful Restart

Ability to enable and disable graceful restart, helper and no GR at all mode functionality at peer level.

So bgp graceful restart can be enabled at modes global BGP level or at per peer level. There are two FSM, one for BGP GR global mode and other for peer per GR.

Default global mode is helper and default peer per mode is inherit from global. If per peer mode is configured, the GR mode of this particular peer will override the global mode.

BGP GR Global Mode Commands

`bgp graceful-restart`

This command will enable BGP graceful restart functionality at the global level.

`bgp graceful-restart disable`

This command will disable both the functionality graceful restart and helper mode.

BGP GR Peer Mode Commands

`neighbor A.B.C.D graceful-restart`
This command will enable BGP graceful restart functionality at the peer level.

`neighbor A.B.C.D graceful-restart-helper`
This command will enable BGP graceful restart helper only functionality at the peer level.

`neighbor A.B.C.D graceful-restart-disable`
This command will disable the entire BGP graceful restart functionality at the peer level.

Long-lived Graceful Restart

Currently, only restarter mode is supported. This capability is advertised only if graceful restart capability is negotiated.

`bgp long-lived-graceful-restart stale-time (1-16777215)`
Specifies the maximum time to wait before purging long-lived stale routes for helper routers.

Default is 0, which means the feature is off by default. Only graceful restart takes into account.

Administrative Shutdown

`bgp shutdown [message MSG...]`
Administrative shutdown of all peers of a bgp instance. Drop all BGP peers, but preserve their configurations. The peers are notified in accordance with RFC 8203 by sending a `NOTIFICATION` message with error code `Cease` and subcode `Administrative Shutdown` prior to terminating connections. This global shutdown is independent of the neighbor shutdown, meaning that individually shut down peers will not be affected by lifting it.

An optional shutdown message *MSG* can be specified.

Networks

`network A.B.C.D/M`
This command adds the announcement network.

```
router bgp 1
 address-family ipv4 unicast
  network 10.0.0.0/8
 exit-address-family
```

This configuration example says that network 10.0.0.0/8 will be announced to all neighbors. Some vendors' routers don't advertise routes if they aren't present in their IGP routing tables; *bgpd* doesn't care about IGP routes when announcing its routes.

`bgp network import-check`
This configuration modifies the behavior of the network statement. If you have this configured the underlying network must exist in the rib. If you have the [no] form configured then BGP will not check for the networks existence in the rib. For versions 7.3 and before frr defaults for datacenter were the network must exist, traditional did not check for existence. For versions 7.4 and beyond both traditional and datacenter the network must exist.

IPv6 Support

neighbor A.B.C.D activate

This configuration modifies whether to enable an address family for a specific neighbor. By default only the IPv4 unicast address family is enabled.

```
router bgp 1
 address-family ipv6 unicast
  neighbor 2001:0DB8::1 activate
  network 2001:0DB8:5009::/64
 exit-address-family
```

This configuration example says that network 2001:0DB8:5009::/64 will be announced and enables the neighbor 2001:0DB8::1 to receive this announcement.

By default, only the IPv4 unicast address family is announced to all neighbors. Using the 'no bgp default ipv4-unicast' configuration overrides this default so that all address families need to be enabled explicitly.

```
router bgp 1
 no bgp default ipv4-unicast
 neighbor 10.10.10.1 remote-as 2
 neighbor 2001:0DB8::1 remote-as 3
 address-family ipv4 unicast
  neighbor 10.10.10.1 activate
  network 192.168.1.0/24
 exit-address-family
 address-family ipv6 unicast
  neighbor 2001:0DB8::1 activate
  network 2001:0DB8:5009::/64
 exit-address-family
```

This configuration demonstrates how the 'no bgp default ipv4-unicast' might be used in a setup with two upstreams where each of the upstreams should only receive either IPv4 or IPv6 announcements.

Using the `bgp default ipv6-unicast` configuration, IPv6 unicast address family is enabled by default for all new neighbors.

Route Aggregation

Route Aggregation-IPv4 Address Family

aggregate-address A.B.C.D/M

This command specifies an aggregate address.

In order to advertise an aggregated prefix, a more specific (longer) prefix MUST exist in the BGP table. For example, if you want to create an `aggregate-address 10.0.0.0/24`, you should make sure you have something like `10.0.0.5/32` or `10.0.0.0/26`, or any other smaller prefix in the BGP table. The routing information table (RIB) is not enough, you have to redistribute them into the BGP table.

aggregate-address A.B.C.D/M route-map NAME

Apply a route-map for an aggregated prefix.

aggregate-address A.B.C.D/M origin <egp|igp|incomplete>

Override ORIGIN for an aggregated prefix.

aggregate-address A.B.C.D/M as-set
This command specifies an aggregate address. Resulting routes include AS set.

aggregate-address A.B.C.D/M summary-only
This command specifies an aggregate address.

Longer prefixes advertisements of more specific routes to all neighbors are suppressed.

aggregate-address A.B.C.D/M matching-MED-only
Configure the aggregated address to only be created when the routes MED match, otherwise no aggregated route will be created.

aggregate-address A.B.C.D/M suppress-map NAME
Similar to *summary-only*, but will only suppress more specific routes that are matched by the selected route-map.

This configuration example sets up an `aggregate-address` under the ipv4 address-family.

```
router bgp 1
 address-family ipv4 unicast
  aggregate-address 10.0.0.0/8
  aggregate-address 20.0.0.0/8 as-set
  aggregate-address 40.0.0.0/8 summary-only
  aggregate-address 50.0.0.0/8 route-map aggr-rmap
 exit-address-family
```

Route Aggregation-IPv6 Address Family

aggregate-address X:X::X:X/M
This command specifies an aggregate address.

aggregate-address X:X::X:X/M route-map NAME
Apply a route-map for an aggregated prefix.

aggregate-address X:X::X:X/M origin <egp|igp|incomplete>
Override ORIGIN for an aggregated prefix.

aggregate-address X:X::X:X/M as-set
This command specifies an aggregate address. Resulting routes include AS set.

aggregate-address X:X::X:X/M summary-only
This command specifies an aggregate address.

Longer prefixes advertisements of more specific routes to all neighbors are suppressed

aggregate-address X:X::X:X/M matching-MED-only
Configure the aggregated address to only be created when the routes MED match, otherwise no aggregated route will be created.

aggregate-address X:X::X:X/M suppress-map NAME
Similar to *summary-only*, but will only suppress more specific routes that are matched by the selected route-map.

This configuration example sets up an `aggregate-address` under the ipv6 address-family.

```
router bgp 1
 address-family ipv6 unicast
  aggregate-address 10::0/64
  aggregate-address 20::0/64 as-set
  aggregate-address 40::0/64 summary-only
```

(continues on next page)

(continued from previous page)

```
 aggregate-address 50::0/64 route-map aggr-rmap
exit-address-family
```

Redistribution

Redistribution configuration should be placed under the `address-family` section for the specific AF to redistribute into. Protocol availability for redistribution is determined by BGP AF; for example, you cannot redistribute OSPFv3 into `address-family ipv4 unicast` as OSPFv3 supports IPv6.

`redistribute <babel|connected|eigrp|isis|kernel|openfabric|ospf|ospf6|rip|ripng|sharp|static|table> [me`

Redistribute routes from other protocols into BGP.

`redistribute vnc-direct`

Redistribute VNC direct (not via zebra) routes to BGP process.

`bgp update-delay MAX-DELAY`

`bgp update-delay MAX-DELAY ESTABLISH-WAIT`

This feature is used to enable read-only mode on BGP process restart or when a BGP process is cleared using 'clear ip bgp *'. Note that this command is configured at the global level and applies to all bgp instances/vrfs. It cannot be used at the same time as the "update-delay" command described below, which is entered in each bgp instance/vrf desired to delay update installation and advertisements. The global and per-vrf approaches to defining update-delay are mutually exclusive.

When applicable, read-only mode would begin as soon as the first peer reaches Established status and a timer for max-delay seconds is started. During this mode BGP doesn't run any best-path or generate any updates to its peers. This mode continues until:

1. All the configured peers, except the shutdown peers, have sent explicit EOR (End-Of-RIB) or an implicit-EOR. The first keep-alive after BGP has reached Established is considered an implicit-EOR. If the establish-wait optional value is given, then BGP will wait for peers to reach established from the beginning of the update-delay till the establish-wait period is over, i.e. the minimum set of established peers for which EOR is expected would be peers established during the establish-wait window, not necessarily all the configured neighbors.
2. max-delay period is over.

On hitting any of the above two conditions, BGP resumes the decision process and generates updates to its peers.

Default max-delay is 0, i.e. the feature is off by default.

`update-delay MAX-DELAY`

`update-delay MAX-DELAY ESTABLISH-WAIT`

This feature is used to enable read-only mode on BGP process restart or when a BGP process is cleared using 'clear ip bgp *'. Note that this command is configured under the specific bgp instance/vrf that the feature is enabled for. It cannot be used at the same time as the global "bgp update-delay" described above, which is entered at the global level and applies to all bgp instances. The global and per-vrf approaches to defining update-delay are mutually exclusive.

When applicable, read-only mode would begin as soon as the first peer reaches Established status and a timer for max-delay seconds is started. During this mode BGP doesn't run any best-path or generate any updates to its peers. This mode continues until:

1. All the configured peers, except the shutdown peers, have sent explicit EOR (End-Of-RIB) or an implicit-EOR. The first keep-alive after BGP has reached Established is considered an implicit-EOR. If the establish-wait optional value is given, then BGP will wait for peers to reach established from the beginning of the

update-delay till the establish-wait period is over, i.e. the minimum set of established peers for which EOR is expected would be peers established during the establish-wait window, not necessarily all the configured neighbors.

2. max-delay period is over.

On hitting any of the above two conditions, BGP resumes the decision process and generates updates to its peers.

Default max-delay is 0, i.e. the feature is off by default.

`table-map ROUTE-MAP-NAME`

This feature is used to apply a route-map on route updates from BGP to Zebra. All the applicable match operations are allowed, such as match on prefix, next-hop, communities, etc. Set operations for this attach-point are limited to metric and next-hop only. Any operation of this feature does not affect BGPs internal RIB.

Supported for ipv4 and ipv6 address families. It works on multi-paths as well, however, metric setting is based on the best-path only.

Peers

Defining Peers

`neighbor PEER remote-as ASN`

Creates a new neighbor whose remote-as is ASN. PEER can be an IPv4 address or an IPv6 address or an interface to use for the connection.

```
router bgp 1
 neighbor 10.0.0.1 remote-as 2
```

In this case my router, in AS-1, is trying to peer with AS-2 at 10.0.0.1.

This command must be the first command used when configuring a neighbor. If the remote-as is not specified, *bgpd* will complain like this:

```
can't find neighbor 10.0.0.1
```

`neighbor PEER remote-as internal`

Create a peer as you would when you specify an ASN, except that if the peers ASN is different than mine as specified under the *`router bgp ASN`* command the connection will be denied.

`neighbor PEER remote-as external`

Create a peer as you would when you specify an ASN, except that if the peers ASN is the same as mine as specified under the *`router bgp ASN`* command the connection will be denied.

`bgp listen range <A.B.C.D/M|X:X::X:X/M> peer-group PGNAME`

Accept connections from any peers in the specified prefix. Configuration from the specified peer-group is used to configure these peers.

Note: When using BGP listen ranges, if the associated peer group has TCP MD5 authentication configured, your kernel must support this on prefixes. On Linux, this support was added in kernel version 4.14. If your kernel does not support this feature you will get a warning in the log file, and the listen range will only accept connections from peers without MD5 configured.

Additionally, we have observed that when using this option at scale (several hundred peers) the kernel may hit its option memory limit. In this situation you will see error messages like:

`bgpd: sockopt_tcp_signature: setsockopt(23): Cannot allocate memory`

In this case you need to increase the value of the sysctl `net.core.optmem_max` to allow the kernel to allocate the necessary option memory.

`bgp listen limit <1-65535>`

Define the maximum number of peers accepted for one BGP instance. This limit is set to 100 by default. Increasing this value will really be possible if more file descriptors are available in the BGP process. This value is defined by the underlying system (ulimit value), and can be overridden by *–limit-fds*. More information is available in chapter (*Common Invocation Options*).

`coalesce-time (0-4294967295)`

The time in milliseconds that BGP will delay before deciding what peers can be put into an update-group together in order to generate a single update for them. The default time is 1000.

Configuring Peers

`neighbor PEER shutdown [message MSG...] [rtt (1-65535) [count (1-255)]]`

Shutdown the peer. We can delete the neighbor's configuration by `no neighbor PEER remote-as ASN` but all configuration of the neighbor will be deleted. When you want to preserve the configuration, but want to drop the BGP peer, use this syntax.

Optionally you can specify a shutdown message *MSG*.

Also, you can specify optionally `rtt` in milliseconds to automatically shutdown the peer if round-trip-time becomes higher than defined.

Additional `count` parameter is the number of keepalive messages to count before shutdown the peer if round-trip-time becomes higher than defined.

`neighbor PEER disable-connected-check`

Allow peerings between directly connected eBGP peers using loopback addresses.

`neighbor PEER disable-link-bw-encoding-ieee`

By default bandwidth in extended communities is carried encoded as IEEE floating-point format, which is according to the draft.

Older versions have the implementation where extended community bandwidth value is carried encoded as uint32. To enable backward compatibility we need to disable IEEE floating-point encoding option per-peer.

`neighbor PEER extended-optional-parameters`

Force Extended Optional Parameters Length format to be used for OPEN messages.

By default, it's disabled. If the standard optional parameters length is higher than one-octet (255), then extended format is enabled automatically.

For testing purposes, extended format can be enabled with this command.

`neighbor PEER ebgp-multihop`

Specifying `ebgp-multihop` allows sessions with eBGP neighbors to establish when they are multiple hops away. When the neighbor is not directly connected and this knob is not enabled, the session will not establish.

If the peer's IP address is not in the RIB and is reachable via the default route, then you have to enable `ip nht resolve-via-default`.

`neighbor PEER description ...`

Set description of the peer.

`neighbor PEER interface IFNAME`

When you connect to a BGP peer over an IPv6 link-local address, you have to specify the IFNAME of the interface used for the connection. To specify IPv4 session addresses, see the `neighbor PEER update-source` command below.

`neighbor PEER interface remote-as <internal|external|ASN>`
Configure an unnumbered BGP peer. `PEER` should be an interface name. The session will be established via IPv6 link locals. Use `internal` for iBGP and `external` for eBGP sessions, or specify an ASN if you wish.

`neighbor PEER next-hop-self [force]`
This command specifies an announced route's nexthop as being equivalent to the address of the bgp router if it is learned via eBGP. This will also bypass third-party next-hops in favor of the local bgp address. If the optional keyword `force` is specified the modification is done also for routes learned via iBGP.

`neighbor PEER attribute-unchanged [{as-path|next-hop|med}]`
This command specifies attributes to be left unchanged for advertisements sent to a peer. Use this to leave the next-hop unchanged in ipv6 configurations, as the route-map directive to leave the next-hop unchanged is only available for ipv4.

`neighbor PEER update-source <IFNAME|ADDRESS>`
Specify the IPv4 source address to use for the BGP session to this neighbour, may be specified as either an IPv4 address directly or as an interface name (in which case the *zebra* daemon MUST be running in order for *bgpd* to be able to retrieve interface state).

```
router bgp 64555
 neighbor foo update-source 192.168.0.1
 neighbor bar update-source lo0
```

`neighbor PEER default-originate [route-map WORD]`
bgpd's default is to not announce the default route (0.0.0.0/0) even if it is in routing table. When you want to announce default routes to the peer, use this command.

If `route-map` keyword is specified, then the default route will be originated only if route-map conditions are met. For example, announce the default route only if `10.10.10.10/32` route exists and set an arbitrary community for a default route.

```
router bgp 64555
 address-family ipv4 unicast
  neighbor 192.168.255.1 default-originate route-map default
!
ip prefix-list p1 seq 5 permit 10.10.10.10/32
!
route-map default permit 10
 match ip address prefix-list p1
 set community 123:123
!
```

`neighbor PEER port PORT`

`neighbor PEER password PASSWORD`
Set a MD5 password to be used with the tcp socket that is being used to connect to the remote peer. Please note if you are using this command with a large number of peers on linux you should consider modifying the *net.core.optmem_max* sysctl to a larger value to avoid out of memory errors from the linux kernel.

`neighbor PEER send-community`

`neighbor PEER weight WEIGHT`
This command specifies a default *weight* value for the neighbor's routes.

`neighbor PEER maximum-prefix NUMBER [force]`
Sets a maximum number of prefixes we can receive from a given peer. If this number is exceeded, the BGP

session will be destroyed.

In practice, it is generally preferable to use a prefix-list to limit what prefixes are received from the peer instead of using this knob. Tearing down the BGP session when a limit is exceeded is far more destructive than merely rejecting undesired prefixes. The prefix-list method is also much more granular and offers much smarter matching criterion than number of received prefixes, making it more suited to implementing policy.

If `force` is set, then ALL prefixes are counted for maximum instead of accepted only. This is useful for cases where an inbound filter is applied, but you want maximum-prefix to act on ALL (including filtered) prefixes. This option requires *soft-reconfiguration inbound* to be enabled for the peer.

`neighbor PEER maximum-prefix-out NUMBER`

Sets a maximum number of prefixes we can send to a given peer.

Since sent prefix count is managed by update-groups, this option creates a separate update-group for outgoing updates.

`neighbor PEER local-as AS-NUMBER [no-prepend] [replace-as]`

Specify an alternate AS for this BGP process when interacting with the specified peer. With no modifiers, the specified local-as is prepended to the received AS_PATH when receiving routing updates from the peer, and prepended to the outgoing AS_PATH (after the process local AS) when transmitting local routes to the peer.

If the no-prepend attribute is specified, then the supplied local-as is not prepended to the received AS_PATH.

If the replace-as attribute is specified, then only the supplied local-as is prepended to the AS_PATH when transmitting local-route updates to this peer.

Note that replace-as can only be specified if no-prepend is.

This command is only allowed for eBGP peers.

`neighbor <A.B.C.D|X:X::X:X|WORD> as-override`

Override AS number of the originating router with the local AS number.

Usually this configuration is used in PEs (Provider Edge) to replace the incoming customer AS number so the connected CE (Customer Edge) can use the same AS number as the other customer sites. This allows customers of the provider network to use the same AS number across their sites.

This command is only allowed for eBGP peers.

`neighbor <A.B.C.D|X:X::X:X|WORD> allowas-in [<(1-10)|origin>]`

Accept incoming routes with AS path containing AS number with the same value as the current system AS.

This is used when you want to use the same AS number in your sites, but you can't connect them directly. This is an alternative to *neighbor WORD as-override*.

The parameter *(1-10)* configures the amount of accepted occurrences of the system AS number in AS path.

The parameter *origin* configures BGP to only accept routes originated with the same AS number as the system.

This command is only allowed for eBGP peers.

`neighbor <A.B.C.D|X:X::X:X|WORD> addpath-tx-all-paths`

Configure BGP to send all known paths to neighbor in order to preserve multi path capabilities inside a network.

`neighbor <A.B.C.D|X:X::X:X|WORD> addpath-tx-bestpath-per-AS`

Configure BGP to send best known paths to neighbor in order to preserve multi path capabilities inside a network.

`neighbor <A.B.C.D|X:X::X:X|WORD> disable-addpath-rx`

Do not accept additional paths from this neighbor.

`neighbor PEER ttl-security hops NUMBER`

This command enforces Generalized TTL Security Mechanism (GTSM), as specified in RFC 5082. With this command, only neighbors that are the specified number of hops away will be allowed to become neighbors. This command is mutually exclusive with *ebgp-multihop*.

`neighbor PEER capability extended-nexthop`
Allow bgp to negotiate the extended-nexthop capability with it's peer. If you are peering over a v6 LL address then this capability is turned on automatically. If you are peering over a v6 Global Address then turning on this command will allow BGP to install v4 routes with v6 nexthops if you do not have v4 configured on interfaces.

`neighbor <A.B.C.D|X:X::X:X|WORD> accept-own`
Enable handling of self-originated VPN routes containing `accept-own` community.

This feature allows you to handle self-originated VPN routes, which a BGP speaker receives from a route-reflector. A 'self-originated' route is one that was originally advertised by the speaker itself. As per **RFC 4271**, a BGP speaker rejects advertisements that originated the speaker itself. However, the BGP ACCEPT_OWN mechanism enables a router to accept the prefixes it has advertised, when reflected from a route-reflector that modifies certain attributes of the prefix.

A special community called `accept-own` is attached to the prefix by the route-reflector, which is a signal to the receiving router to bypass the ORIGINATOR_ID and NEXTHOP/MP_REACH_NLRI check.

Default: disabled.

`neighbor <A.B.C.D|X:X::X:X|WORD> path-attribute discard (1-255)...`
Drops specified path attributes from BGP UPDATE messages from the specified neighbor.

If you do not want specific attributes, you can drop them using this command, and let the BGP proceed by ignoring those attributes.

`neighbor <A.B.C.D|X:X::X:X|WORD> path-attribute treat-as-withdraw (1-255)...`
Received BGP UPDATES that contain specified path attributes are treat-as-withdraw. If there is an existing prefix in the BGP routing table, it will be removed.

`neighbor <A.B.C.D|X:X::X:X|WORD> graceful-shutdown`
Mark all routes from this neighbor as less preferred by setting `graceful-shutdown` community, and local-preference to 0.

`bgp fast-external-failover`
This command causes bgp to take down ebgp peers immediately when a link flaps. *bgp fast-external-failover* is the default and will not be displayed as part of a *show run*. The no form of the command turns off this ability.

`bgp default ipv4-unicast`
This command allows the user to specify that the IPv4 Unicast address family is turned on by default or not. This command defaults to on and is not displayed. The *no bgp default ipv4-unicast* form of the command is displayed.

`bgp default ipv4-multicast`
This command allows the user to specify that the IPv4 Multicast address family is turned on by default or not. This command defaults to off and is not displayed. The *bgp default ipv4-multicast* form of the command is displayed.

`bgp default ipv4-vpn`
This command allows the user to specify that the IPv4 MPLS VPN address family is turned on by default or not. This command defaults to off and is not displayed. The *bgp default ipv4-vpn* form of the command is displayed.

`bgp default ipv4-flowspec`
This command allows the user to specify that the IPv4 Flowspec address family is turned on by default or not. This command defaults to off and is not displayed. The *bgp default ipv4-flowspec* form of the command is displayed.

`bgp default ipv6-unicast`
This command allows the user to specify that the IPv6 Unicast address family is turned on by default or not. This command defaults to off and is not displayed. The *bgp default ipv6-unicast* form of the command is displayed.

`bgp default ipv6-multicast`
: This command allows the user to specify that the IPv6 Multicast address family is turned on by default or not. This command defaults to off and is not displayed. The *bgp default ipv6-multicast* form of the command is displayed.

`bgp default ipv6-vpn`
: This command allows the user to specify that the IPv6 MPLS VPN address family is turned on by default or not. This command defaults to off and is not displayed. The *bgp default ipv6-vpn* form of the command is displayed.

`bgp default ipv6-flowspec`
: This command allows the user to specify that the IPv6 Flowspec address family is turned on by default or not. This command defaults to off and is not displayed. The *bgp default ipv6-flowspec* form of the command is displayed.

`bgp default l2vpn-evpn`
: This command allows the user to specify that the L2VPN EVPN address family is turned on by default or not. This command defaults to off and is not displayed. The *bgp default l2vpn-evpn* form of the command is displayed.

`bgp default show-hostname`
: This command shows the hostname of the peer in certain BGP commands outputs. It's easier to troubleshoot if you have a number of BGP peers.

`bgp default show-nexthop-hostname`
: This command shows the hostname of the next-hop in certain BGP commands outputs. It's easier to troubleshoot if you have a number of BGP peers and a number of routes to check.

`neighbor PEER advertisement-interval (0-600)`
: Setup the minimum route advertisement interval(mrai) for the peer in question. This number is between 0 and 600 seconds, with the default advertisement interval being 0.

`neighbor PEER timers (0-65535) (0-65535)`
: Set keepalive and hold timers for a neighbor. The first value is keepalive and the second is hold time.

`neighbor PEER timers connect (1-65535)`
: Set connect timer for a neighbor. The connect timer controls how long BGP waits between connection attempts to a neighbor.

`neighbor PEER timers delayopen (1-240)`
: This command allows the user enable the *RFC 4271 <https://tools.ietf.org/html/rfc4271/>* DelayOpenTimer with the specified interval or disable it with the negating command for the peer. By default, the DelayOpenTimer is disabled. The timer interval may be set to a duration of 1 to 240 seconds.

`bgp minimum-holdtime (1-65535)`
: This command allows user to prevent session establishment with BGP peers with lower holdtime less than configured minimum holdtime. When this command is not set, minimum holdtime does not work.

`bgp tcp-keepalive (1-65535) (1-65535) (1-30)`
: This command allows user to configure TCP keepalive with new BGP peers. Each parameter respectively stands for TCP keepalive idle timer (seconds), interval (seconds), and maximum probes. By default, TCP keepalive is disabled.

Displaying Information about Peers

`show bgp <afi> <safi> neighbors WORD bestpath-routes [detail] [json] [wide]`

For the given neighbor, WORD, that is specified list the routes selected by BGP as having the best path.

If `detail` option is specified, the detailed version of all routes will be displayed. The same format as `show [ip] bgp [afi] [safi] PREFIX` will be used, but for the whole table of received, advertised or filtered prefixes.

If `json` option is specified, output is displayed in JSON format.

If `wide` option is specified, then the prefix table's width is increased to fully display the prefix and the nexthop.

Peer Filtering

`neighbor PEER distribute-list NAME [in|out]`

This command specifies a distribute-list for the peer. *direct* is `in` or `out`.

`neighbor PEER prefix-list NAME [in|out]`

`neighbor PEER filter-list NAME [in|out]`

`neighbor PEER route-map NAME [in|out]`

Apply a route-map on the neighbor. *direct* must be *in* or *out*.

`bgp route-reflector allow-outbound-policy`

By default, attribute modification via route-map policy out is not reflected on reflected routes. This option allows the modifications to be reflected as well. Once enabled, it affects all reflected routes.

`neighbor PEER sender-as-path-loop-detection`

Enable the detection of sender side AS path loops and filter the bad routes before they are sent.

This setting is disabled by default.

Peer Groups

Peer groups are used to help improve scaling by generating the same update information to all members of a peer group. Note that this means that the routes generated by a member of a peer group will be sent back to that originating peer with the originator identifier attribute set to indicated the originating peer. All peers not associated with a specific peer group are treated as belonging to a default peer group, and will share updates.

`neighbor WORD peer-group`

This command defines a new peer group.

`neighbor PEER peer-group PGNAME`

This command bind specific peer to peer group WORD.

`neighbor PEER solo`

This command is used to indicate that routes advertised by the peer should not be reflected back to the peer. This command only is only meaningful when there is a single peer defined in the peer-group.

`show [ip] bgp peer-group [json]`

This command displays configured BGP peer-groups.

```
exit1-debian-9# show bgp peer-group

BGP peer-group test1, remote AS 65001
Peer-group type is external
```

(continues on next page)

(continued from previous page)

```
Configured address-families: IPv4 Unicast; IPv6 Unicast;
1 IPv4 listen range(s)
   192.168.100.0/24
2 IPv6 listen range(s)
   2001:db8:1::/64
   2001:db8:2::/64
Peer-group members:
   192.168.200.1  Active
   2001:db8::1  Active

BGP peer-group test2
Peer-group type is external
Configured address-families: IPv4 Unicast;
```

Optional `json` parameter is used to display JSON output.

```
{
  "test1":{
    "remoteAs":65001,
    "type":"external",
    "addressFamiliesConfigured":[
      "IPv4 Unicast",
      "IPv6 Unicast"
    ],
    "dynamicRanges":{
      "IPv4":{
        "count":1,
        "ranges":[
          "192.168.100.0\/24"
        ]
      },
      "IPv6":{
        "count":2,
        "ranges":[
          "2001:db8:1::\/64",
          "2001:db8:2::\/64"
        ]
      }
    },
    "members":{
      "192.168.200.1":{
        "status":"Active"
      },
      "2001:db8::1":{
        "status":"Active"
      }
    }
  },
  "test2":{
     "type":"external",
     "addressFamiliesConfigured":[
       "IPv4 Unicast"
```

(continues on next page)

(continued from previous page)

```
      ]
    }
  }
```

Capability Negotiation

`neighbor PEER strict-capability-match`

Strictly compares remote capabilities and local capabilities. If capabilities are different, send Unsupported Capability error then reset connection.

You may want to disable sending Capability Negotiation OPEN message optional parameter to the peer when remote peer does not implement Capability Negotiation. Please use *dont-capability-negotiate* command to disable the feature.

`neighbor PEER dont-capability-negotiate`

Suppress sending Capability Negotiation as OPEN message optional parameter to the peer. This command only affects the peer is configured other than IPv4 unicast configuration.

When remote peer does not have capability negotiation feature, remote peer will not send any capabilities at all. In that case, bgp configures the peer with configured capabilities.

You may prefer locally configured capabilities more than the negotiated capabilities even though remote peer sends capabilities. If the peer is configured by *override-capability*, *bgpd* ignores received capabilities then override negotiated capabilities with configured values.

Additionally the operator should be reminded that this feature fundamentally disables the ability to use widely deployed BGP features. BGP unnumbered, hostname support, AS4, Addpath, Route Refresh, ORF, Dynamic Capabilities, and graceful restart.

`neighbor PEER override-capability`

Override the result of Capability Negotiation with local configuration. Ignore remote peer's capability value.

`neighbor PEER capability software-version`

Send the software version in the BGP OPEN message to the neighbor. This is very useful in environments with a large amount of peers with different versions of FRR or any other vendor.

Disabled by default.

`neighbor PEER aigp`

Send and receive AIGP attribute for this neighbor. This is valid only for eBGP neighbors.

Disabled by default. iBGP neighbors have this option enabled implicitly.

AS Path Access Lists

AS path access list is user defined AS path.

`bgp as-path access-list WORD [seq (0-4294967295)] permit|deny LINE`

This command defines a new AS path access list.

`show bgp as-path-access-list [json]`

Display all BGP AS Path access lists.

If the `json` option is specified, output is displayed in JSON format.

`show bgp as-path-access-list WORD [json]`

Display the specified BGP AS Path access list.

If the `json` option is specified, output is displayed in JSON format.

Bogon ASN filter policy configuration example

```
bgp as-path access-list 99 permit _0_
bgp as-path access-list 99 permit _23456_
bgp as-path access-list 99 permit _1310[0-6][0-9]_|_13107[0-1]_
bgp as-path access-list 99 seq 20 permit ^65
```

Using AS Path in Route Map

`match as-path WORD`
For a given as-path, WORD, match it on the BGP as-path given for the prefix and if it matches do normal route-map actions. The no form of the command removes this match from the route-map.

`set as-path prepend AS-PATH`
Prepend the given string of AS numbers to the AS_PATH of the BGP path's NLRI. The no form of this command removes this set operation from the route-map.

`set as-path prepend last-as NUM`
Prepend the existing last AS number (the leftmost ASN) to the AS_PATH. The no form of this command removes this set operation from the route-map.

`set as-path replace <any|ASN>`
Replace a specific AS number to local AS number. `any` replaces each AS number in the AS-PATH with the local AS number.

Communities Attribute

The BGP communities attribute is widely used for implementing policy routing. Network operators can manipulate BGP communities attribute based on their network policy. BGP communities attribute is defined in **RFC 1997** and **RFC 1998**. It is an optional transitive attribute, therefore local policy can travel through different autonomous system.

The communities attribute is a set of communities values. Each community value is 4 octet long. The following format is used to define the community value.

`AS:VAL` This format represents 4 octet communities value. `AS` is high order 2 octet in digit format. `VAL` is low order 2 octet in digit format. This format is useful to define AS oriented policy value. For example, `7675:80` can be used when AS 7675 wants to pass local policy value 80 to neighboring peer.

`graceful-shutdown` `graceful-shutdown` represents well-known communities value `GRACEFUL_SHUTDOWN` `0xFFFF0000` `65535:0`. **RFC 8326** implements the purpose Graceful BGP Session Shutdown to reduce the amount of lost traffic when taking BGP sessions down for maintenance. The use of the community needs to be supported from your peers side to actually have any effect.

`accept-own` `accept-own` represents well-known communities value `ACCEPT_OWN` `0xFFFF0001` `65535:1`. **RFC 7611** implements a way to signal to a router to accept routes with a local nexthop address. This can be the case when doing policing and having traffic having a nexthop located in another VRF but still local interface to the router. It is recommended to read the RFC for full details.

`route-filter-translated-v4` `route-filter-translated-v4` represents well-known communities value `ROUTE_FILTER_TRANSLATED_v4` `0xFFFF0002` `65535:2`.

`route-filter-v4` `route-filter-v4` represents well-known communities value `ROUTE_FILTER_v4` `0xFFFF0003` `65535:3`.

`route-filter-translated-v6` `route-filter-translated-v6` represents well-known communities value `ROUTE_FILTER_TRANSLATED_v6` `0xFFFF0004` `65535:4`.

`route-filter-v6` `route-filter-v6` represents well-known communities value `ROUTE_FILTER_v6` `0xFFFF0005` `65535:5`.

`llgr-stale` `llgr-stale` represents well-known communities value `LLGR_STALE` `0xFFFF0006` `65535:6`. Assigned and intended only for use with routers supporting the Long-lived Graceful Restart Capability as described in [Draft-IETF-uttaro-idr-bgp-persistence]. Routers receiving routes with this community may (depending on implementation) choose allow to reject or modify routes on the presence or absence of this community.

`no-llgr` `no-llgr` represents well-known communities value `NO_LLGR` `0xFFFF0007` `65535:7`. Assigned and intended only for use with routers supporting the Long-lived Graceful Restart Capability as described in [Draft-IETF-uttaro-idr-bgp-persistence]. Routers receiving routes with this community may (depending on implementation) choose allow to reject or modify routes on the presence or absence of this community.

`accept-own-nexthop` `accept-own-nexthop` represents well-known communities value `accept-own-nexthop` `0xFFFF0008` `65535:8`. [Draft-IETF-agrewal-idr-accept-own-nexthop] describes how to tag and label VPN routes to be able to send traffic between VRFs via an internal layer 2 domain on the same PE device. Refer to [Draft-IETF-agrewal-idr-accept-own-nexthop] for full details.

`blackhole` `blackhole` represents well-known communities value `BLACKHOLE` `0xFFFF029A` `65535:666`. **RFC 7999** documents sending prefixes to EBGP peers and upstream for the purpose of blackholing traffic. Prefixes tagged with the this community should normally not be re-advertised from neighbors of the originating network. Upon receiving `BLACKHOLE` community from a BGP speaker, `NO_ADVERTISE` community is added automatically.

`no-export` `no-export` represents well-known communities value `NO_EXPORT` `0xFFFFFF01`. All routes carry this value must not be advertised to outside a BGP confederation boundary. If neighboring BGP peer is part of BGP confederation, the peer is considered as inside a BGP confederation boundary, so the route will be announced to the peer.

`no-advertise` `no-advertise` represents well-known communities value `NO_ADVERTISE` `0xFFFFFF02`. All routes carry this value must not be advertise to other BGP peers.

`local-AS` `local-AS` represents well-known communities value `NO_EXPORT_SUBCONFED` `0xFFFFFF03`. All routes carry this value must not be advertised to external BGP peers. Even if the neighboring router is part of confederation, it is considered as external BGP peer, so the route will not be announced to the peer.

`no-peer` `no-peer` represents well-known communities value `NOPEER` `0xFFFFFF04` `65535:65284`. **RFC 3765** is used to communicate to another network how the originating network want the prefix propagated.

When the communities attribute is received duplicate community values in the attribute are ignored and value is sorted in numerical order.

Community Lists

Community lists are user defined lists of community attribute values. These lists can be used for matching or manipulating the communities attribute in UPDATE messages.

There are two types of community list:

standard This type accepts an explicit value for the attribute.

expanded This type accepts a regular expression. Because the regex must be interpreted on each use expanded community lists are slower than standard lists.

`bgp community-list standard NAME permit|deny COMMUNITY`
This command defines a new standard community list. `COMMUNITY` is communities value. The `COMMUNITY` is compiled into community structure. We can define multiple community list under same name. In that case match will happen user defined order. Once the community list matches to communities attribute in BGP updates it

return permit or deny by the community list definition. When there is no matched entry, deny will be returned. When `COMMUNITY` is empty it matches to any routes.

`bgp community-list expanded NAME permit|deny COMMUNITY`
This command defines a new expanded community list. `COMMUNITY` is a string expression of communities attribute. `COMMUNITY` can be a regular expression (*BGP Regular Expressions*) to match the communities attribute in BGP updates. The expanded community is only used to filter, not *set* actions.

Deprecated since version 5.0: It is recommended to use the more explicit versions of this command.

`bgp community-list NAME permit|deny COMMUNITY`
When the community list type is not specified, the community list type is automatically detected. If `COMMUNITY` can be compiled into communities attribute, the community list is defined as a standard community list. Otherwise it is defined as an expanded community list. This feature is left for backward compatibility. Use of this feature is not recommended.

Note that all community lists share the same namespace, so it's not necessary to specify `standard` or `expanded`; these modifiers are purely aesthetic.

`show bgp community-list [NAME detail]`
Displays community list information. When `NAME` is specified the specified community list's information is shown.

```
# show bgp community-list
Named Community standard list CLIST
permit 7675:80 7675:100 no-export
deny internet
  Named Community expanded list EXPAND
permit :

  # show bgp community-list CLIST detail
  Named Community standard list CLIST
permit 7675:80 7675:100 no-export
deny internet
```

Numbered Community Lists

When number is used for BGP community list name, the number has special meanings. Community list number in the range from 1 to 99 is standard community list. Community list number in the range from 100 to 500 is expanded community list. These community lists are called as numbered community lists. On the other hand normal community lists is called as named community lists.

`bgp community-list (1-99) permit|deny COMMUNITY`
This command defines a new community list. The argument to (1-99) defines the list identifier.

`bgp community-list (100-500) permit|deny COMMUNITY`
This command defines a new expanded community list. The argument to (100-500) defines the list identifier.

Community alias

BGP community aliases are useful to quickly identify what communities are set for a specific prefix in a human-readable format. Especially handy for a huge amount of communities. Accurately defined aliases can help you faster spot things on the wire.

`bgp community alias NAME ALIAS`

This command creates an alias name for a community that will be used later in various CLI outputs in a human-readable format.

```
~# vtysh -c 'show run' | grep 'bgp community alias'
bgp community alias 65001:14 community-1
bgp community alias 65001:123:1 lcommunity-1

~# vtysh -c 'show ip bgp 172.16.16.1/32'
BGP routing table entry for 172.16.16.1/32, version 21
Paths: (2 available, best #2, table default)
  Advertised to non peer-group peers:
  65030
    192.168.0.2 from 192.168.0.2 (172.16.16.1)
      Origin incomplete, metric 0, valid, external, best (Neighbor IP)
      Community: 65001:12 65001:13 community-1 65001:65534
      Large Community: lcommunity-1 65001:123:2
      Last update: Fri Apr 16 12:51:27 2021
```

`show bgp [afi] [safi] [all] alias WORD [wide|json]`

Display prefixes with matching BGP community alias.

Using Communities in Route Maps

In *Route Maps* we can match on or set the BGP communities attribute. Using this feature network operator can implement their network policy based on BGP communities attribute.

The following commands can be used in route maps:

`match alias WORD`

This command performs match to BGP updates using community alias WORD. When the one of BGP communities value match to the one of community alias value in community alias, it is match.

`match community WORD exact-match [exact-match]`

This command perform match to BGP updates using community list WORD. When the one of BGP communities value match to the one of communities value in community list, it is match. When *exact-match* keyword is specified, match happen only when BGP updates have completely same communities value specified in the community list.

`set community <none|COMMUNITY> additive`

This command sets the community value in BGP updates. If the attribute is already configured, the newly provided value replaces the old one unless the `additive` keyword is specified, in which case the new value is appended to the existing value.

If `none` is specified as the community value, the communities attribute is not sent.

It is not possible to set an expanded community list.

`set comm-list WORD delete`

This command remove communities value from BGP communities attribute. The `word` is community list name. When BGP route's communities value matches to the community list `word`, the communities value is removed.

When all of communities value is removed eventually, the BGP update's communities attribute is completely removed.

Example Configuration

The following configuration is exemplary of the most typical usage of BGP communities attribute. In the example, AS 7675 provides an upstream Internet connection to AS 100. When the following configuration exists in AS 7675, the network operator of AS 100 can set local preference in AS 7675 network by setting BGP communities attribute to the updates.

```
router bgp 7675
 neighbor 192.168.0.1 remote-as 100
 address-family ipv4 unicast
  neighbor 192.168.0.1 route-map RMAP in
 exit-address-family
!
bgp community-list 70 permit 7675:70
bgp community-list 80 permit 7675:80
bgp community-list 90 permit 7675:90
!
route-map RMAP permit 10
 match community 70
 set local-preference 70
!
route-map RMAP permit 20
 match community 80
 set local-preference 80
!
route-map RMAP permit 30
 match community 90
 set local-preference 90
```

The following configuration announces `10.0.0.0/8` from AS 100 to AS 7675. The route has communities value `7675:80` so when above configuration exists in AS 7675, the announced routes' local preference value will be set to 80.

```
router bgp 100
 network 10.0.0.0/8
 neighbor 192.168.0.2 remote-as 7675
 address-family ipv4 unicast
  neighbor 192.168.0.2 route-map RMAP out
 exit-address-family
!
ip prefix-list PLIST permit 10.0.0.0/8
!
route-map RMAP permit 10
 match ip address prefix-list PLIST
 set community 7675:80
```

The following configuration is an example of BGP route filtering using communities attribute. This configuration only permit BGP routes which has BGP communities value (`0:80` and `0:90`) or `0:100`. The network operator can set special internal communities value at BGP border router, then limit the BGP route announcements into the internal network.

```
router bgp 7675
 neighbor 192.168.0.1 remote-as 100
 address-family ipv4 unicast
  neighbor 192.168.0.1 route-map RMAP in
 exit-address-family
!
bgp community-list 1 permit 0:80 0:90
bgp community-list 1 permit 0:100
!
route-map RMAP permit in
 match community 1
```

The following example filters BGP routes which have a community value of `1:1`. When there is no match community-list returns `deny`. To avoid filtering all routes, a `permit` line is set at the end of the community-list.

```
router bgp 7675
 neighbor 192.168.0.1 remote-as 100
 address-family ipv4 unicast
  neighbor 192.168.0.1 route-map RMAP in
 exit-address-family
!
bgp community-list standard FILTER deny 1:1
bgp community-list standard FILTER permit
!
route-map RMAP permit 10
 match community FILTER
```

The following configuration is an example of communities value deletion. With this configuration the community values `100:1` and `100:2` are removed from BGP updates. For communities value deletion, only `permit` community-list is used. `deny` community-list is ignored.

```
router bgp 7675
 neighbor 192.168.0.1 remote-as 100
 address-family ipv4 unicast
  neighbor 192.168.0.1 route-map RMAP in
 exit-address-family
!
bgp community-list standard DEL permit 100:1 100:2
!
route-map RMAP permit 10
 set comm-list DEL delete
```

Extended Communities Attribute

BGP extended communities attribute is introduced with MPLS VPN/BGP technology. MPLS VPN/BGP expands capability of network infrastructure to provide VPN functionality. At the same time it requires a new framework for policy routing. With BGP Extended Communities Attribute we can use Route Target or Site of Origin for implementing network policy for MPLS VPN/BGP.

BGP Extended Communities Attribute is similar to BGP Communities Attribute. It is an optional transitive attribute. BGP Extended Communities Attribute can carry multiple Extended Community value. Each Extended Community value is eight octet length.

BGP Extended Communities Attribute provides an extended range compared with BGP Communities Attribute. Adding to that there is a type field in each value to provides community space structure.

There are two format to define Extended Community value. One is AS based format the other is IP address based format.

`AS:VAL` This is a format to define AS based Extended Community value. `AS` part is 2 octets Global Administrator subfield in Extended Community value. `VAL` part is 4 octets Local Administrator subfield. `7675:100` represents AS 7675 policy value 100.

`IP-Address:VAL` This is a format to define IP address based Extended Community value. `IP-Address` part is 4 octets Global Administrator subfield. `VAL` part is 2 octets Local Administrator subfield.

Extended Community Lists

`bgp extcommunity-list standard NAME permit|deny EXTCOMMUNITY`
This command defines a new standard extcommunity-list. *extcommunity* is extended communities value. The *extcommunity* is compiled into extended community structure. We can define multiple extcommunity-list under same name. In that case match will happen user defined order. Once the extcommunity-list matches to extended communities attribute in BGP updates it return permit or deny based upon the extcommunity-list definition. When there is no matched entry, deny will be returned. When *extcommunity* is empty it matches to any routes.

`bgp extcommunity-list expanded NAME permit|deny LINE`
This command defines a new expanded extcommunity-list. *line* is a string expression of extended communities attribute. *line* can be a regular expression (*BGP Regular Expressions*) to match an extended communities attribute in BGP updates.

Note that all extended community lists shares a single name space, so it's not necessary to specify their type when creating or destroying them.

`show bgp extcommunity-list [NAME detail]`
This command displays current extcommunity-list information. When *name* is specified the community list's information is shown.

BGP Extended Communities in Route Map

`match extcommunity WORD`

`set extcommunity none`
This command resets the extended community value in BGP updates. If the attribute is already configured or received from the peer, the attribute is discarded and set to none. This is useful if you need to strip incoming extended communities.

`set extcommunity rt EXTCOMMUNITY`
This command sets Route Target value.

`set extcommunity nt EXTCOMMUNITY`
This command sets Node Target value.

If the receiving BGP router supports Node Target Extended Communities, it will install the route with the community that contains it's own local BGP Identifier. Otherwise, it's not installed.

`set extcommunity soo EXTCOMMUNITY`
This command sets Site of Origin value.

`set extcommunity bandwidth <(1-25600) | cumulative | num-multipaths> [non-transitive]`
This command sets the BGP link-bandwidth extended community for the prefix (best path) for which it is applied. The link-bandwidth can be specified as an `explicit value` (specified in Mbps), or the router can be told to

use the `cumulative bandwidth` of all multipaths for the prefix or to compute it based on the `number of multipaths`. The link bandwidth extended community is encoded as `transitive` unless the set command explicitly configures it as `non-transitive`.

See also:

Weighted ECMP using BGP link bandwidth

Note that the extended expanded community is only used for *match* rule, not for *set* actions.

Large Communities Attribute

The BGP Large Communities attribute was introduced in Feb 2017 with **RFC 8092**.

The BGP Large Communities Attribute is similar to the BGP Communities Attribute except that it has 3 components instead of two and each of which are 4 octets in length. Large Communities bring additional functionality and convenience over traditional communities, specifically the fact that the `GLOBAL` part below is now 4 octets wide allowing seamless use in networks using 4-byte ASNs.

`GLOBAL:LOCAL1:LOCAL2` This is the format to define Large Community values. Referencing **RFC 8195** the values are commonly referred to as follows:

- The `GLOBAL` part is a 4 octet Global Administrator field, commonly used as the operators AS number.
- The `LOCAL1` part is a 4 octet Local Data Part 1 subfield referred to as a function.
- The `LOCAL2` part is a 4 octet Local Data Part 2 field and referred to as the parameter subfield.

As an example, `65551:1:10` represents AS 65551 function 1 and parameter 10. The referenced RFC above gives some guidelines on recommended usage.

Large Community Lists

Two types of large community lists are supported, namely *standard* and *expanded*.

`bgp large-community-list standard NAME permit|deny LARGE-COMMUNITY`

This command defines a new standard large-community-list. *large-community* is the Large Community value. We can add multiple large communities under same name. In that case the match will happen in the user defined order. Once the large-community-list matches the Large Communities attribute in BGP updates it will return permit or deny based upon the large-community-list definition. When there is no matched entry, a deny will be returned. When *large-community* is empty it matches any routes.

`bgp large-community-list expanded NAME permit|deny LINE`

This command defines a new expanded large-community-list. Where *line* is a string matching expression, it will be compared to the entire Large Communities attribute as a string, with each large-community in order from lowest to highest. *line* can also be a regular expression which matches this Large Community attribute.

Note that all community lists share the same namespace, so it's not necessary to specify `standard` or `expanded`; these modifiers are purely aesthetic.

`show bgp large-community-list`

`show bgp large-community-list NAME detail`

This command display current large-community-list information. When *name* is specified the community list information is shown.

`show ip bgp large-community-info`

This command displays the current large communities in use.

Large Communities in Route Map

match large-community LINE [exact-match]

Where *line* can be a simple string to match, or a regular expression. It is very important to note that this match occurs on the entire large-community string as a whole, where each large-community is ordered from lowest to highest. When *exact-match* keyword is specified, match happen only when BGP updates have completely same large communities value specified in the large community list.

set large-community LARGE-COMMUNITY

set large-community LARGE-COMMUNITY LARGE-COMMUNITY

set large-community LARGE-COMMUNITY additive

These commands are used for setting large-community values. The first command will overwrite any large-communities currently present. The second specifies two large-communities, which overwrites the current large-community list. The third will add a large-community value without overwriting other values. Multiple large-community values can be specified.

Note that the large expanded community is only used for *match* rule, not for *set* actions.

BGP Roles and Only to Customers

BGP roles are defined in **RFC 9234** and provide an easy way to route leaks prevention, detection and mitigation.

To enable its mechanics, you must set your local role to reflect your type of peering relationship with your neighbor. Possible values of `LOCAL-ROLE` are:

- provider
- rs-server
- rs-client
- customer
- peer

The local Role value is negotiated with the new BGP Role capability with a built-in check of the corresponding value. In case of mismatch the new OPEN Roles Mismatch Notification <2, 11> would be sent.

The correct Role pairs are:

- Provider - Customer
- Peer - Peer
- RS-Server - RS-Client

```
~# vtysh -c 'show bgp neighbor' | grep 'Role'
  Local Role: customer
  Neighbor Role: provider
    Role: advertised and received
```

If strict-mode is set BGP session won't become established until BGP neighbor set local Role on its side. This configuration parameter is defined in **RFC 9234** and used to enforce corresponding configuration at your counter-part side. Default value - disabled.

Routes that sent from provider, rs-server, or peer local-role (or if received by customer, rs-clinet, or peer local-role) will be marked with a new Only to Customer (OTC) attribute.

Routes with this attribute can only be sent to your neighbor if your local-role is provider or rs-server. Routes with this attribute can be received only if your local-role is customer or rs-client.

In case of peer-peer relationship routes can be received only if OTC value is equal to your neighbor AS number.

All these rules with OTC help to detect and mitigate route leaks and happened automatically if local-role is set.

`neighbor PEER local-role LOCAL-ROLE [strict-mode]`

> This command set your local-role to `LOCAL-ROLE`: <provider|rs-server|rs-client|customer|peer>.
>
> This role helps to detect and prevent route leaks.
>
> If `strict-mode` is set, your neighbor must send you Capability with the value of his role (by setting local-role on his side). Otherwise, a Role Mismatch Notification will be sent.

Labeled unicast

bgpd supports labeled information, as per **RFC 3107**.

`bgp labeled-unicast <explicit-null|ipv4-explicit-null|ipv6-explicit-null>`

By default, locally advertised prefixes use the *implicit-null* label to encode in the outgoing NLRI. The following command uses the *explicit-null* label value for all the BGP instances.

L3VPN VRFs

bgpd supports L3VPN (Layer 3 Virtual Private Networks) VRFs (Virtual Routing and Forwarding) for IPv4 **RFC 4364** and IPv6 **RFC 4659**. L3VPN routes, and their associated VRF MPLS labels, can be distributed to VPN SAFI neighbors in the *default*, i.e., non VRF, BGP instance. VRF MPLS labels are reached using *core* MPLS labels which are distributed using LDP or BGP labeled unicast. *bgpd* also supports inter-VRF route leaking.

L3VPN over GRE interfaces

In MPLS-VPN or SRv6-VPN, an L3VPN next-hop entry requires that the path chosen respectively contains a labelled path or a valid SID IPv6 address. Otherwise the L3VPN entry will not be installed. It is possible to ignore that check when the path chosen by the next-hop uses a GRE interface, and there is a route-map configured at inbound side of ipv4-vpn or ipv6-vpn address family with following syntax:

`set l3vpn next-hop encapsulation gre`

The incoming BGP L3VPN entry is accepted, provided that the next hop of the L3VPN entry uses a path that takes the GRE tunnel as outgoing interface. The remote endpoint should be configured just behind the GRE tunnel; remote device configuration may vary depending whether it acts at edge endpoint or not: in any case, the expectation is that incoming MPLS traffic received at this endpoint should be considered as a valid path for L3VPN.

VRF Route Leaking

BGP routes may be leaked (i.e. copied) between a unicast VRF RIB and the VPN SAFI RIB of the default VRF for use in MPLS-based L3VPNs. Unicast routes may also be leaked between any VRFs (including the unicast RIB of the default BGP instanced). A shortcut syntax is also available for specifying leaking from one VRF to another VRF using the default instance's VPN RIB as the intermediary. A common application of the VRF-VRF feature is to connect a customer's private routing domain to a provider's VPN service. Leaking is configured from the point of view of an individual VRF: `import` refers to routes leaked from VPN to a unicast VRF, whereas `export` refers to routes leaked from a unicast VRF to VPN.

Required parameters

Routes exported from a unicast VRF to the VPN RIB must be augmented by two parameters:

- an RD (Route Distinguisher)
- an RTLIST (Route-target List)

Configuration for these exported routes must, at a minimum, specify these two parameters.

Routes imported from the VPN RIB to a unicast VRF are selected according to their RTLISTs. Routes whose RTLIST contains at least one route-target in common with the configured import RTLIST are leaked. Configuration for these imported routes must specify an RTLIST to be matched.

The RD, which carries no semantic value, is intended to make the route unique in the VPN RIB among all routes of its prefix that originate from all the customers and sites that are attached to the provider's VPN service. Accordingly, each site of each customer is typically assigned an RD that is unique across the entire provider network.

The RTLIST is a set of route-target extended community values whose purpose is to specify route-leaking policy. Typically, a customer is assigned a single route-target value for import and export to be used at all customer sites. This configuration specifies a simple topology wherein a customer has a single routing domain which is shared across all its sites. More complex routing topologies are possible through use of additional route-targets to augment the leaking of sets of routes in various ways.

When using the shortcut syntax for vrf-to-vrf leaking, the RD and RT are auto-derived.

General configuration

Configuration of route leaking between a unicast VRF RIB and the VPN SAFI RIB of the default VRF is accomplished via commands in the context of a VRF address-family:

`rd vpn export AS:NN|IP:nn`
Specifies the route distinguisher to be added to a route exported from the current unicast VRF to VPN.

`rt vpn import|export|both RTLIST...`
Specifies the route-target list to be attached to a route (export) or the route-target list to match against (import) when exporting/importing between the current unicast VRF and VPN.

The RTLIST is a space-separated list of route-targets, which are BGP extended community values as described in *Extended Communities Attribute*.

`label vpn export allocation-mode per-vrf|per-nexthop`
Select how labels are allocated in the given VRF. By default, the *per-vrf* mode is selected, and one label is used for all prefixes from the VRF. The *per-nexthop* will use a unique label for all prefixes that are reachable via the same nexthop.

`label vpn export (0..1048575)|auto`
Enables an MPLS label to be attached to a route exported from the current unicast VRF to VPN. If the value specified is `auto`, the label value is automatically assigned from a pool maintained by the Zebra daemon. If Zebra is not running, or if this command is not configured, automatic label assignment will not complete, which will block corresponding route export.

`nexthop vpn export A.B.C.D|X:X::X:X`
Specifies an optional nexthop value to be assigned to a route exported from the current unicast VRF to VPN. If left unspecified, the nexthop will be set to 0.0.0.0 or 0:0::0:0 (self).

`route-map vpn import|export MAP`
Specifies an optional route-map to be applied to routes imported or exported between the current unicast VRF and VPN.

import|export vpn
Enables import or export of routes between the current unicast VRF and VPN.

import vrf VRFNAME
Shortcut syntax for specifying automatic leaking from vrf VRFNAME to the current VRF using the VPN RIB as intermediary. The RD and RT are auto derived and should not be specified explicitly for either the source or destination VRF's.

This shortcut syntax mode is not compatible with the explicit *import vpn* and *export vpn* statements for the two VRF's involved. The CLI will disallow attempts to configure incompatible leaking modes.

bgp retain route-target all

It is possible to retain or not VPN prefixes that are not imported by local VRF configuration. This can be done via the following command in the context of the global VPNv4/VPNv6 family. This command defaults to on and is not displayed. The *no bgp retain route-target all* form of the command is displayed.

neighbor <A.B.C.D|X:X::X:X|WORD> soo EXTCOMMUNITY

Without this command, SoO extended community attribute is configured using an inbound route map that sets the SoO value during the update process. With the introduction of the new BGP per-neighbor Site-of-Origin (SoO) feature, two new commands configured in sub-modes under router configuration mode simplify the SoO value configuration.

If we configure SoO per neighbor at PEs, the SoO community is automatically added for all routes from the CPEs. Routes are validated and prevented from being sent back to the same CPE (e.g.: multi-site). This is especially needed when using `as-override` or `allowas-in` to prevent routing loops.

mpls bgp forwarding

It is possible to permit BGP install VPN prefixes without transport labels, by issuing the following command under the interface configuration context. This configuration will install VPN prefixes originated from an e-bgp session, and with the next-hop directly connected.

L3VPN SRv6

segment-routing srv6
Use SRv6 backend with BGP L3VPN, and go to its configuration node.

locator NAME
Specify the SRv6 locator to be used for SRv6 L3VPN. The Locator name must be set in zebra, but user can set it in any order.

General configuration

Configuration of the SRv6 SID used to advertise a L3VPN for both IPv4 and IPv6 is accomplished via the following command in the context of a VRF:

sid vpn per-vrf export (1..1048575)|auto
Enables a SRv6 SID to be attached to a route exported from the current unicast VRF to VPN. A single SID is used for both IPv4 and IPv6 address families. If you want to set a SID for only IPv4 address family or IPv6 address family, you need to use the command `sid vpn export (1..1048575)|auto` in the context of an address-family. If the value specified is `auto`, the SID value is automatically assigned from a pool maintained by the Zebra daemon. If Zebra is not running, or if this command is not configured, automatic SID assignment will not complete, which will block corresponding route export.

Ethernet Virtual Network - EVPN

Note: When using EVPN features and if you have a large number of hosts, make sure to adjust the size of the arp neighbor cache to avoid neighbor table overflow and/or excessive garbage collection. On Linux, the size of the table and garbage collection frequency can be controlled via the following sysctl configurations:

```
net.ipv4.neigh.default.gc_thresh1
net.ipv4.neigh.default.gc_thresh2
net.ipv4.neigh.default.gc_thresh3

net.ipv6.neigh.default.gc_thresh1
net.ipv6.neigh.default.gc_thresh2
net.ipv6.neigh.default.gc_thresh3
```

For more information, see `man 7 arp`.

Enabling EVPN

EVPN should be enabled on the BGP instance corresponding to the VRF acting as the underlay for the VXLAN tunneling. In most circumstances this will be the default VRF. The command to enable EVPN for a BGP instance is `advertise-all-vni` which lives under `address-family l2vpn evpn`:

```
router bgp 65001
 !
 address-family l2vpn evpn
  advertise-all-vni
```

A more comprehensive configuration example can be found in the *EVPN* page.

EVPN L3 Route-Targets

`route-target <import|export|both> <RTLIST|auto>`

Modify the route-target set for EVPN advertised type-2/type-5 routes. RTLIST is a list of any of matching `(A.B.C.D:MN|EF:OPQR|GHJK:MN|*:OPQR|*:MN)` where `*` indicates wildcard matching for the AS number. It will be set to match any AS number. This is useful in datacenter deployments with Downstream VNI. `auto` is used to retain the autoconfigure that is default behavior for L3 RTs.

EVPN advertise-PIP

In a EVPN symmetric routing MLAG deployment, all EVPN routes advertised with anycast-IP as next-hop IP and anycast MAC as the Router MAC (RMAC - in BGP EVPN Extended-Community). EVPN picks up the next-hop IP from the VxLAN interface's local tunnel IP and the RMAC is obtained from the MAC of the L3VNI's SVI interface. Note: Next-hop IP is used for EVPN routes whether symmetric routing is deployed or not but the RMAC is only relevant for symmetric routing scenario.

Current behavior is not ideal for Prefix (type-5) and self (type-2) routes. This is because the traffic from remote VTEPs routed sub optimally if they land on the system where the route does not belong.

The advertise-pip feature advertises Prefix (type-5) and self (type-2) routes with system's individual (primary) IP as the next-hop and individual (system) MAC as Router-MAC (RMAC), while leaving the behavior unchanged for other EVPN routes.

To support this feature there needs to have ability to co-exist a (system-MAC, system-IP) pair with a (anycast-MAC, anycast-IP) pair with the ability to terminate VxLAN-encapsulated packets received for either pair on the same L3VNI (i.e associated VLAN). This capability is needed per tenant VRF instance.

To derive the system-MAC and the anycast MAC, there must be a separate/additional MAC-VLAN interface corresponding to L3VNI's SVI. The SVI interface's MAC address can be interpreted as system-MAC and MAC-VLAN interface's MAC as anycast MAC.

To derive system-IP and anycast-IP, the default BGP instance's router-id is used as system-IP and the VxLAN interface's local tunnel IP as the anycast-IP.

User has an option to configure the system-IP and/or system-MAC value if the auto derived value is not preferred.

Note: By default, advertise-pip feature is enabled and user has an option to disable the feature via configuration CLI. Once the feature is disabled under bgp vrf instance or MAC-VLAN interface is not configured, all the routes follow the same behavior of using same next-hop and RMAC values.

`advertise-pip [ip <addr> [mac <addr>]]`

Enables or disables advertise-pip feature, specify system-IP and/or system-MAC parameters.

EVPN advertise-svi-ip

Typically, the SVI IP address is reused on VTEPs across multiple racks. However, if you have unique SVI IP addresses that you want to be reachable you can use the advertise-svi-ip option. This option advertises the SVI IP/MAC address as a type-2 route and eliminates the need for any flooding over VXLAN to reach the IP from a remote VTEP.

`advertise-svi-ip`

Note that you should not enable both the advertise-svi-ip and the advertise-default-gw at the same time.

EVPN Overlay Index Gateway IP

RFC https://datatracker.ietf.org/doc/html/rfc9136 explains the use of overlay indexes for recursive route resolution for EVPN type-5 route.

We support gateway IP overlay index. A gateway IP, advertised with EVPN prefix route, is used to find an EVPN MAC/IP route with its IP field same as the gateway IP. This MAC/IP entry provides the nexthop VTEP and the tunnel information required for the VxLAN encapsulation.

Functionality:

```
.      +---------+ BGP  +---------+ BGP  +---------+      +---------+
 SN1   |         | IPv4 |         | EVPN |         |      |         |
======+ Host1   +-------+   PE1   +-------+   PE2   +-------+  Host2  +
       |         |      |         |      |         |      |         |
       +---------+      +---------+      +---------+      +---------+
```

Consider above topology where prefix SN1 is connected behind host1. Host1 advertises SN1 to PE1 over BGP IPv4 session. PE1 advertises SN1 to PE2 using EVPN type-5 route with host1 IP as the gateway IP. PE1 also advertises Host1 MAC/IP as type-2 route which is used to resolve host1 gateway IP.

PE2 receives this type-5 route and imports it into the vrf based on route targets. BGP prefix imported into the vrf uses gateway IP as its BGP nexthop. This route is installed into zebra if following conditions are satisfied:

1. Gateway IP nexthop is L3 reachable.
2. PE2 has received EVPN type-2 route with IP field set to gateway IP.

Topology requirements:

1. This feature is supported for asymmetric routing model only. While sending packets to SN1, ingress PE (PE2) performs routing and egress PE (PE1) performs only bridging.
2. This feature supports only traditional(non vlan-aware) bridge model. Bridge interface associated with L2VNI is an L3 interface. i.e., this interface is configured with an address in the L2VNI subnet. Note that the gateway IP should also have an address in the same subnet.
3. As this feature works in asymmetric routing model, all L2VNIs and corresponding VxLAN and bridge interfaces should be present at all the PEs.
4. L3VNI configuration is required to generate and import EVPN type-5 routes. L3VNI VxLAN and bridge interfaces also should be present.

A PE can use one of the following two mechanisms to advertise an EVPN type-5 route with gateway IP.

1. CLI to add gateway IP while generating EVPN type-5 route from a BGP IPv4/IPv6 prefix:

`advertise <ipv4|ipv6> unicast [gateway-ip]`

When this CLI is configured for a BGP vrf under L2VPN EVPN address family, EVPN type-5 routes are generated for BGP prefixes in the vrf. Nexthop of the BGP prefix becomes the gateway IP of the corresponding type-5 route.

If the above command is configured without the "gateway-ip" keyword, type-5 routes are generated without overlay index.

2. Add gateway IP to EVPN type-5 route using a route-map:

`set evpn gateway-ip <ipv4|ipv6> <addr>`

When route-map with above set clause is applied as outbound policy in BGP, it will set the gateway-ip in EVPN type-5 NLRI.

Example configuration:

```
router bgp 100
 neighbor 192.168.0.1 remote-as 101
 !
 address-family ipv4 l2vpn evpn
  neighbor 192.168.0.1 route-map RMAP out
 exit-address-family
!
route-map RMAP permit 10
 set evpn gateway-ip 10.0.0.1
 set evpn gateway-ip 10::1
```

A PE that receives a type-5 route with gateway IP overlay index should have "enable-resolve-overlay-index" configuration enabled to recursively resolve the overlay index nexthop and install the prefix into zebra.

`enable-resolve-overlay-index`

Example configuration:

```
router bgp 65001
 bgp router-id 192.168.100.1
 no bgp ebgp-requires-policy
 neighbor 10.0.1.2 remote-as 65002
 !
 address-family l2vpn evpn
  neighbor 10.0.1.2 activate
```

(continues on next page)

(continued from previous page)

```
  advertise-all-vni
  enable-resolve-overlay-index
 exit-address-family
!
```

EVPN Multihoming

All-Active Multihoming is used for redundancy and load sharing. Servers are attached to two or more PEs and the links are bonded (link-aggregation). This group of server links is referred to as an Ethernet Segment.

Ethernet Segments

An Ethernet Segment can be configured by specifying a system-MAC and a local discriminator or a complete ESINAME against the bond interface on the PE (via zebra) -

`evpn mh es-id <(1-16777215)|ESINAME>`

`evpn mh es-sys-mac X:X:X:X:X:X`

The sys-mac and local discriminator are used for generating a 10-byte, Type-3 Ethernet Segment ID. ESINAME is a 10-byte, Type-0 Ethernet Segment ID - “00:AA:BB:CC:DD:EE:FF:GG:HH:II”.

Type-1 (EAD-per-ES and EAD-per-EVI) routes are used to advertise the locally attached ESs and to learn off remote ESs in the network. Local Type-2/MAC-IP routes are also advertised with a destination ESI allowing for MAC-IP syncing between Ethernet Segment peers. Reference: RFC 7432, RFC 8365

EVPN-MH is intended as a replacement for MLAG or Anycast VTEPs. In multihoming each PE has an unique VTEP address which requires the introduction of a new dataplane construct, MAC-ECMP. Here a MAC/FDB entry can point to a list of remote PEs/VTEPs.

BUM handling

Type-4 (ESR) routes are used for Designated Forwarder (DF) election. DFs forward BUM traffic received via the overlay network. This implementation uses a preference based DF election specified by draft-ietf-bess-evpn-pref-df. The DF preference is configurable per-ES (via zebra) -

`evpn mh es-df-pref (1-16777215)`

BUM traffic is rxed via the overlay by all PEs attached to a server but only the DF can forward the de-capsulated traffic to the access port. To accommodate that non-DF filters are installed in the dataplane to drop the traffic.

Similarly traffic received from ES peers via the overlay cannot be forwarded to the server. This is split-horizon-filtering with local bias.

Knobs for interop

Some vendors do not send EAD-per-EVI routes. To interop with them we need to relax the dependency on EAD-per-EVI routes and activate a remote ES-PE based on just the EAD-per-ES route.

Note that by default we advertise and expect EAD-per-EVI routes.

`disable-ead-evi-rx`

`disable-ead-evi-tx`

Fast failover

As the primary purpose of EVPN-MH is redundancy keeping the failover efficient is a recurring theme in the implementation. Following sub-features have been introduced for the express purpose of efficient ES failovers.

- Layer-2 Nexthop Groups and MAC-ECMP via L2NHG.
- Host routes (for symmetric IRB) via L3NHG. On dataplanes that support layer3 nexthop groups the feature can be turned on via the following BGP config -

`use-es-l3nhg`

- Local ES (MAC/Neigh) failover via ES-redirect. On dataplanes that do not have support for ES-redirect the feature can be turned off via the following zebra config -

`evpn mh redirect-off`

Uplink/Core tracking

When all the underlay links go down the PE no longer has access to the VxLAN +overlay. To prevent blackholing of traffic the server/ES links are protodowned on the PE. A link can be setup for uplink tracking via the following zebra configuration -

`evpn mh uplink`

Proxy advertisements

To handle hitless upgrades support for proxy advertisement has been added as specified by draft-rbickhart-evpn-ip-mac-proxy-adv. This allows a PE (say PE1) to proxy advertise a MAC-IP rxed from an ES peer (say PE2). When the ES peer (PE2) goes down PE1 continues to advertise hosts learnt from PE2 for a holdtime during which it attempts to establish local reachability of the host. This holdtime is configurable via the following zebra commands -

`evpn mh neigh-holdtime (0-86400)`

`evpn mh mac-holdtime (0-86400)`

Startup delay

When a switch is rebooted we wait for a brief period to allow the underlay and EVPN network to converge before enabling the ESs. For this duration the ES bonds are held protodown. The startup delay is configurable via the following zebra command -

evpn mh startup-delay (0-3600)

EAD-per-ES fragmentation

The EAD-per-ES route carries the EVI route targets for all the broadcast domains associated with the ES. Depending on the EVI scale the EAD-per-ES route maybe fragmented.

The number of EVIs per-EAD route can be configured via the following BGP command -

[no] ead-es-frag evi-limit (1-1000)

Sample Configuration

```
!
router bgp 5556
 !
 address-family l2vpn evpn
  ead-es-frag evi-limit 200
 exit-address-family
 !
!
```

EAD-per-ES route-target

The EAD-per-ES route by default carries all the EVI route targets. Depending on EVI scale that can result in route fragmentation. In some cases it maybe necessary to avoid this fragmentation and that can be done via the following workaround - 1. Configure a single supplementary BD per-tenant VRF. This SBD needs to be provisioned on all EVPN PEs associated with the tenant-VRF. 2. Config the SBD's RT as the EAD-per-ES route's export RT.

Sample Configuration

```
!
router bgp 5556
 !
 address-family l2vpn evpn
  ead-es-route-target export 5556:1001
  ead-es-route-target export 5556:1004
  ead-es-route-target export 5556:1008
 exit-address-family
!
```

Support with VRF network namespace backend

It is possible to separate overlay networks contained in VXLAN interfaces from underlay networks by using VRFs. VRF-lite and VRF-netns backends can be used for that. In the latter case, it is necessary to set both bridge and vxlan interface in the same network namespace, as below example illustrates:

```
# linux shell
ip netns add vrf1
ip link add name vxlan101 type vxlan id 101 dstport 4789 dev eth0 local 10.1.1.1
ip link set dev vxlan101 netns vrf1
ip netns exec vrf1 ip link set dev lo up
ip netns exec vrf1 brctl addbr bridge101
ip netns exec vrf1 brctl addif bridge101 vxlan101
```

This makes it possible to separate not only layer 3 networks like VRF-lite networks. Also, VRF netns based make possible to separate layer 2 networks on separate VRF instances.

BGP Conditional Advertisement

The BGP conditional advertisement feature uses the `non-exist-map` or the `exist-map` and the `advertise-map` keywords of the neighbor advertise-map command in order to track routes by the route prefix.

`non-exist-map`

1. If a route prefix is not present in the output of non-exist-map command, then advertise the route specified by the advertise-map command.
2. If a route prefix is present in the output of non-exist-map command, then do not advertise the route specified by the addvertise-map command.

`exist-map`

1. If a route prefix is present in the output of exist-map command, then advertise the route specified by the advertise-map command.
2. If a route prefix is not present in the output of exist-map command, then do not advertise the route specified by the advertise-map command.

This feature is useful when some prefixes are advertised to one of its peers only if the information from the other peer is not present (due to failure in peering session or partial reachability etc).

The conditional BGP announcements are sent in addition to the normal announcements that a BGP router sends to its peer.

The conditional advertisement process is triggered by the BGP scanner process, which runs every 60 by default. This means that the maximum time for the conditional advertisement to take effect is the value of the process timer.

As an optimization, while the process always runs on each timer expiry, it determines whether or not the conditional advertisement policy or the routing table has changed; if neither have changed, no processing is necessary and the scanner exits early.

`neighbor A.B.C.D advertise-map NAME [exist-map|non-exist-map] NAME`

This command enables BGP scanner process to monitor routes specified by exist-map or non-exist-map command in BGP table and conditionally advertises the routes specified by advertise-map command.

`bgp conditional-advertisement timer (5-240)`

Set the period to rerun the conditional advertisement scanner process. The default is 60 seconds.

Sample Configuration

```
interface enp0s9
 ip address 10.10.10.2/24
!
interface enp0s10
 ip address 10.10.20.2/24
!
interface lo
 ip address 203.0.113.1/32
!
router bgp 2
 bgp log-neighbor-changes
 no bgp ebgp-requires-policy
 neighbor 10.10.10.1 remote-as 1
 neighbor 10.10.20.3 remote-as 3
 !
 address-family ipv4 unicast
  neighbor 10.10.10.1 soft-reconfiguration inbound
  neighbor 10.10.20.3 soft-reconfiguration inbound
  neighbor 10.10.20.3 advertise-map ADV-MAP non-exist-map EXIST-MAP
 exit-address-family
!
ip prefix-list DEFAULT seq 5 permit 192.0.2.5/32
ip prefix-list DEFAULT seq 10 permit 192.0.2.1/32
ip prefix-list EXIST seq 5 permit 10.10.10.10/32
ip prefix-list DEFAULT-ROUTE seq 5 permit 0.0.0.0/0
ip prefix-list IP1 seq 5 permit 10.139.224.0/20
!
bgp community-list standard DC-ROUTES seq 5 permit 64952:3008
bgp community-list standard DC-ROUTES seq 10 permit 64671:501
bgp community-list standard DC-ROUTES seq 15 permit 64950:3009
bgp community-list standard DEFAULT-ROUTE seq 5 permit 65013:200
!
route-map ADV-MAP permit 10
 match ip address prefix-list IP1
!
route-map ADV-MAP permit 20
 match community DC-ROUTES
!
route-map EXIST-MAP permit 10
 match community DEFAULT-ROUTE
 match ip address prefix-list DEFAULT-ROUTE
!
```

Sample Output

When default route is present in R2'2 BGP table, 10.139.224.0/20 and 192.0.2.1/32 are not advertised to R3.

```
Router2# show ip bgp
BGP table version is 20, local router ID is 203.0.113.1, vrf id 0
Default local pref 100, local AS 2
Status codes:  s suppressed, d damped, h history, * valid, > best, = multipath,
               i internal, r RIB-failure, S Stale, R Removed
Nexthop codes: @NNN nexthop's vrf id, < announce-nh-self
Origin codes:  i - IGP, e - EGP, ? - incomplete
RPKI validation codes: V valid, I invalid, N Not found

   Network          Next Hop            Metric LocPrf Weight Path
*> 0.0.0.0/0        10.10.10.1               0             0 1 i
*> 10.139.224.0/20  10.10.10.1               0             0 1 ?
*> 192.0.2.1/32     10.10.10.1               0             0 1 i
*> 192.0.2.5/32     10.10.10.1               0             0 1 i

Displayed  4 routes and 4 total paths
Router2# show ip bgp neighbors 10.10.20.3

!--- Output suppressed.

For address family: IPv4 Unicast
Update group 7, subgroup 7
Packet Queue length 0
Inbound soft reconfiguration allowed
Community attribute sent to this neighbor(all)
Condition NON_EXIST, Condition-map *EXIST-MAP, Advertise-map *ADV-MAP, status: Withdraw
0 accepted prefixes

!--- Output suppressed.

Router2# show ip bgp neighbors 10.10.20.3 advertised-routes
BGP table version is 20, local router ID is 203.0.113.1, vrf id 0
Default local pref 100, local AS 2
Status codes:  s suppressed, d damped, h history, * valid, > best, = multipath,
            i internal, r RIB-failure, S Stale, R Removed
Nexthop codes: @NNN nexthop's vrf id, < announce-nh-self
Origin codes:  i - IGP, e - EGP, ? - incomplete
RPKI validation codes: V valid, I invalid, N Not found

   Network          Next Hop            Metric LocPrf Weight Path
*> 0.0.0.0/0        0.0.0.0                                0 1 i
*> 192.0.2.5/32     0.0.0.0                                0 1 i

Total number of prefixes 2
```

When default route is not present in R2'2 BGP table, 10.139.224.0/20 and 192.0.2.1/32 are advertised to R3.

```
Router2# show ip bgp
BGP table version is 21, local router ID is 203.0.113.1, vrf id 0
```

(continues on next page)

(continued from previous page)

```
Default local pref 100, local AS 2
Status codes:  s suppressed, d damped, h history, * valid, > best, = multipath,
               i internal, r RIB-failure, S Stale, R Removed
Nexthop codes: @NNN nexthop's vrf id, < announce-nh-self
Origin codes:  i - IGP, e - EGP, ? - incomplete
RPKI validation codes: V valid, I invalid, N Not found

   Network          Next Hop            Metric LocPrf Weight Path
*> 10.139.224.0/20  10.10.10.1               0             0 1 ?
*> 192.0.2.1/32     10.10.10.1               0             0 1 i
*> 192.0.2.5/32     10.10.10.1               0             0 1 i

Displayed  3 routes and 3 total paths

Router2# show ip bgp neighbors 10.10.20.3

!--- Output suppressed.

For address family: IPv4 Unicast
Update group 7, subgroup 7
Packet Queue length 0
Inbound soft reconfiguration allowed
Community attribute sent to this neighbor(all)
Condition NON_EXIST, Condition-map *EXIST-MAP, Advertise-map *ADV-MAP, status: Advertise
0 accepted prefixes

!--- Output suppressed.

Router2# show ip bgp neighbors 10.10.20.3 advertised-routes
BGP table version is 21, local router ID is 203.0.113.1, vrf id 0
Default local pref 100, local AS 2
Status codes:  s suppressed, d damped, h history, * valid, > best, = multipath,
               i internal, r RIB-failure, S Stale, R Removed
Nexthop codes: @NNN nexthop's vrf id, < announce-nh-self
Origin codes:  i - IGP, e - EGP, ? - incomplete
RPKI validation codes: V valid, I invalid, N Not found

   Network          Next Hop            Metric LocPrf Weight Path
*> 10.139.224.0/20  0.0.0.0                                0 1 ?
*> 192.0.2.1/32     0.0.0.0                                0 1 i
*> 192.0.2.5/32     0.0.0.0                                0 1 i

Total number of prefixes 3
Router2#
```

Debugging

`show debug`
: Show all enabled debugs.

`show bgp listeners`
: Display Listen sockets and the vrf that created them. Useful for debugging of when listen is not working and this is considered a developer debug statement.

`debug bgp allow-martian`
: Enable or disable BGP accepting martian nexthops from a peer. Please note this is not an actual debug command and this command is also being deprecated and will be removed soon. The new command is *`bgp allow-martian-nexthop`*

`debug bgp bfd`
: Enable or disable debugging for BFD events. This will show BFD integration library messages and BGP BFD integration messages that are mostly state transitions and validation problems.

`debug bgp conditional-advertisement`
: Enable or disable debugging of BGP conditional advertisement.

`debug bgp neighbor-events`
: Enable or disable debugging for neighbor events. This provides general information on BGP events such as peer connection / disconnection, session establishment / teardown, and capability negotiation.

`debug bgp updates`
: Enable or disable debugging for BGP updates. This provides information on BGP UPDATE messages transmitted and received between local and remote instances.

`debug bgp keepalives`
: Enable or disable debugging for BGP keepalives. This provides information on BGP KEEPALIVE messages transmitted and received between local and remote instances.

`debug bgp bestpath <A.B.C.D/M|X:X::X:X/M>`
: Enable or disable debugging for bestpath selection on the specified prefix.

`debug bgp nht`
: Enable or disable debugging of BGP nexthop tracking.

`debug bgp update-groups`
: Enable or disable debugging of dynamic update groups. This provides general information on group creation, deletion, join and prune events.

`debug bgp zebra`
: Enable or disable debugging of communications between *bgpd* and *zebra*.

Dumping Messages and Routing Tables

`dump bgp all PATH [INTERVAL]`

`dump bgp all-et PATH [INTERVAL]`
: Dump all BGP packet and events to *path* file. If *interval* is set, a new file will be created for echo *interval* of seconds. The path *path* can be set with date and time formatting (strftime). The type 'all-et' enables support for Extended Timestamp Header (*Packet Binary Dump Format*).

`dump bgp updates PATH [INTERVAL]`

`dump bgp updates-et PATH [INTERVAL]`
: Dump only BGP updates messages to *path* file. If *interval* is set, a new file will be created for echo *interval* of

seconds. The path *path* can be set with date and time formatting (strftime). The type 'updates-et' enables support for Extended Timestamp Header (*Packet Binary Dump Format*).

`dump bgp routes-mrt PATH`

`dump bgp routes-mrt PATH INTERVAL`

Dump whole BGP routing table to *path*. This is heavy process. The path *path* can be set with date and time formatting (strftime). If *interval* is set, a new file will be created for echo *interval* of seconds.

Note: the interval variable can also be set using hours and minutes: 04h20m00.

Other BGP Commands

The following are available in the top level *enable* mode:

`clear bgp \*`

Clear all peers.

`clear bgp ipv4|ipv6 \*`

Clear all peers with this address-family activated.

`clear bgp ipv4|ipv6 unicast \*`

Clear all peers with this address-family and sub-address-family activated.

`clear bgp ipv4|ipv6 PEER`

Clear peers with address of X.X.X.X and this address-family activated.

`clear bgp ipv4|ipv6 unicast PEER`

Clear peer with address of X.X.X.X and this address-family and sub-address-family activated.

`clear bgp ipv4|ipv6 PEER soft|in|out`

Clear peer using soft reconfiguration in this address-family.

`clear bgp ipv4|ipv6 unicast PEER soft|in|out`

Clear peer using soft reconfiguration in this address-family and sub-address-family.

`clear bgp [ipv4|ipv6] [unicast] PEER|\* message-stats`

Clear BGP message statistics for a specified peer or for all peers, optionally filtered by activated address-family and sub-address-family.

The following are available in the `router bgp` mode:

`write-quanta (1-64)`

BGP message Tx I/O is vectored. This means that multiple packets are written to the peer socket at the same time each I/O cycle, in order to minimize system call overhead. This value controls how many are written at a time. Under certain load conditions, reducing this value could make peer traffic less 'bursty'. In practice, leave this settings on the default (64) unless you truly know what you are doing.

`read-quanta (1-10)`

Unlike Tx, BGP Rx traffic is not vectored. Packets are read off the wire one at a time in a loop. This setting controls how many iterations the loop runs for. As with write-quanta, it is best to leave this setting on the default.

The following command is available in `config` mode as well as in the `router bgp` mode:

`bgp graceful-shutdown`

The purpose of this command is to initiate BGP Graceful Shutdown which is described in **RFC 8326**. The use case for this is to minimize or eliminate the amount of traffic loss in a network when a planned maintenance activity such as software upgrade or hardware replacement is to be performed on a router. The feature works by re-announcing routes to eBGP peers with the GRACEFUL_SHUTDOWN community included. Peers are then expected to treat such paths with the lowest preference. This happens automatically on a receiver running FRR;

with other routing protocol stacks, an inbound policy may have to be configured. In FRR, triggering graceful shutdown also results in announcing a LOCAL_PREF of 0 to iBGP peers.

Graceful shutdown can be configured per BGP instance or globally for all of BGP. These two options are mutually exclusive. The no form of the command causes graceful shutdown to be stopped, and routes will be re-announced without the GRACEFUL_SHUTDOWN community and/or with the usual LOCAL_PREF value. Note that if this option is saved to the startup configuration, graceful shutdown will remain in effect across restarts of *bgpd* and will need to be explicitly disabled.

`bgp input-queue-limit (1-4294967295)`

Set the BGP Input Queue limit for all peers when messaging parsing. Increase this only if you have the memory to handle large queues of messages at once.

`bgp output-queue-limit (1-4294967295)`

Set the BGP Output Queue limit for all peers when messaging parsing. Increase this only if you have the memory to handle large queues of messages at once.

3.3.4 Displaying BGP Information

The following four commands display the IPv6 and IPv4 routing tables, depending on whether or not the `ip` keyword is used. Actually, `show ip bgp` command was used on older *Quagga* routing daemon project, while `show bgp` command is the new format. The choice has been done to keep old format with IPv4 routing table, while new format displays IPv6 routing table.

`show ip bgp [all] [wide|json [detail]]`

`show ip bgp A.B.C.D [json]`

`show bgp [all] [wide|json [detail]]`

`show bgp X:X::X:X [json]`

These commands display BGP routes. When no route is specified, the default is to display all BGP routes.

```
BGP table version is 0, local router ID is 10.1.1.1
   Status codes: s suppressed, d damped, h history, * valid, > best, i - internal
   Origin codes: i - IGP, e - EGP, ? - incomplete

Network    Next Hop      Metric LocPrf Weight Path
   \*> 1.1.1.1/32       0.0.0.0      0   32768 i

   Total number of prefixes 1
```

If `wide` option is specified, then the prefix table's width is increased to fully display the prefix and the nexthop.

This is especially handy dealing with IPv6 prefixes and if `[no] bgp default show-nexthop-hostname` is enabled.

If `all` option is specified, `ip` keyword is ignored, show bgp all and show ip bgp all commands display routes for all AFIs and SAFIs.

If `json` option is specified, output is displayed in JSON format.

If `detail` option is specified after `json`, more verbose JSON output will be displayed.

Some other commands provide additional options for filtering the output.

`show [ip] bgp regexp LINE`

This command displays BGP routes using AS path regular expression (*BGP Regular Expressions*).

show [ip] bgp [all] summary [wide] [json]

Show a bgp peer summary for the specified address family.

The old command structure `show ip bgp` may be removed in the future and should no longer be used. In order to reach the other BGP routing tables other than the IPv6 routing table given by `show bgp`, the new command structure is extended with `show bgp [afi] [safi]`.

`wide` option gives more output like `LocalAS` and extended `Desc` to 64 characters.

```
exit1# show ip bgp summary wide

IPv4 Unicast Summary (VRF default):
BGP router identifier 192.168.100.1, local AS number 65534 vrf-id 0
BGP table version 3
RIB entries 5, using 920 bytes of memory
Peers 1, using 27 KiB of memory

Neighbor        V         AS    LocalAS   MsgRcvd   MsgSent   TblVer  InQ OutQ␣
↪ Up/Down State/PfxRcd   PfxSnt Desc
192.168.0.2     4      65030        123        15        22        0    0    0␣
↪00:07:00            0        1 us-east1-rs1.frrouting.org

Total number of neighbors 1
exit1#
```

If PfxRcd and/or PfxSnt is shown as `(Policy)`, that means that the EBGP default policy is turned on, but you don't have any filters applied for incoming/outgoing directions.

See also:

Require policy on EBGP

show bgp [afi] [safi] [all] [wide|json]

show bgp vrfs [<VRFNAME$vrf_name>] [json]

The command displays all bgp vrf instances basic info like router-id, configured and established neighbors, evpn related basic info like l3vni, router-mac, vxlan-interface. User can get that information as JSON format when `json` keyword at the end of cli is presented.

```
torc-11# show bgp vrfs
Type  Id     routerId          #PeersCfg  #PeersEstb  Name
             L3-VNI            RouterMAC              Interface
DFLT  0      17.0.0.6          3          3           default
             0                 00:00:00:00:00:00      unknown
 VRF  21     17.0.0.6          0          0           sym_1
             8888              34:11:12:22:22:01      vlan4034_l3
 VRF  32     17.0.0.6          0          0           sym_2
             8889              34:11:12:22:22:01      vlan4035_l3

Total number of VRFs (including default): 3
```

show bgp [<ipv4|ipv6> <unicast|multicast|vpn|labeled-unicast|flowspec> | l2vpn evpn]

These commands display BGP routes for the specific routing table indicated by the selected afi and the selected safi. If no afi and no safi value is given, the command falls back to the default IPv6 routing table.

show bgp l2vpn evpn route [type <macip|2|multicast|3|es|4|prefix|5>]

EVPN prefixes can also be filtered by EVPN route type.

show bgp l2vpn evpn route [detail] [type <ead|1|macip|2|multicast|3|es|4|prefix|5>] self-originate [jso

Display self-originated EVPN prefixes which can also be filtered by EVPN route type.

show bgp vni <all|VNI> [vtep VTEP] [type <ead|1|macip|2|multicast|3>] [<detail|json>]

Display per-VNI EVPN routing table in bgp. Filter route-type, vtep, or VNI.

show bgp [afi] [safi] [all] summary [json]

Show a bgp peer summary for the specified address family, and subsequent address-family.

show bgp [afi] [safi] [all] summary failed [json]

Show a bgp peer summary for peers that are not successfully exchanging routes for the specified address family, and subsequent address-family.

show bgp [afi] [safi] [all] summary established [json]

Show a bgp peer summary for peers that are successfully exchanging routes for the specified address family, and subsequent address-family.

show bgp [afi] [safi] [all] summary neighbor [PEER] [json]

Show a bgp summary for the specified peer, address family, and subsequent address-family. The neighbor filter can be used in combination with the failed, established filters.

show bgp [afi] [safi] [all] summary remote-as <internal|external|ASN> [json]

Show a bgp peer summary for the specified remote-as ASN or type (`internal` for iBGP and `external` for eBGP sessions), address family, and subsequent address-family. The remote-as filter can be used in combination with the failed, established filters.

show bgp [afi] [safi] [all] summary terse [json]

Shorten the output. Do not show the following information about the BGP instances: the number of RIB entries, the table version and the used memory. The `terse` option can be used in combination with the remote-as, neighbor, failed and established filters, and with the `wide` option as well.

show bgp [afi] [safi] [neighbor [PEER] [routes|advertised-routes|received-routes] [<A.B.C.D/M|X:X::X:X/M> | detail] [json]

This command shows information on a specific BGP peer of the relevant afi and safi selected.

The `routes` keyword displays only routes in this address-family's BGP table that were received by this peer and accepted by inbound policy.

The `advertised-routes` keyword displays only the routes in this address-family's BGP table that were permitted by outbound policy and advertised to to this peer.

The `received-routes` keyword displays all routes belonging to this address-family (prior to inbound policy) that were received by this peer.

If a specific prefix is specified, the detailed version of that prefix will be displayed.

If `detail` option is specified, the detailed version of all routes will be displayed. The same format as `show [ip] bgp [afi] [safi] PREFIX` will be used, but for the whole table of received, advertised or filtered prefixes.

If `json` option is specified, output is displayed in JSON format.

show bgp [<view|vrf> VIEWVRFNAME] [afi] [safi] neighbors PEER received prefix-filter [json]

Display Address Prefix ORFs received from this peer.

show bgp [afi] [safi] [all] dampening dampened-paths [wide|json]

Display paths suppressed due to dampening of the selected afi and safi selected.

show bgp [afi] [safi] [all] dampening flap-statistics [wide|json]

Display flap statistics of routes of the selected afi and safi selected.

show bgp [afi] [safi] [all] dampening parameters [json]

Display details of configured dampening parameters of the selected afi and safi.

If the `json` option is specified, output is displayed in JSON format.

show bgp [afi] [safi] [all] version (1-4294967295) [wide|json]
Display prefixes with matching version numbers. The version number and above having prefixes will be listed here.

It helps to identify which prefixes were installed at some point.

Here is an example of how to check what prefixes were installed starting with an arbitrary version:

```
# vtysh -c 'show bgp ipv4 unicast json' | jq '.tableVersion'
9
# vtysh -c 'show ip bgp version 9 json' | jq -r '.routes | keys[]'
192.168.3.0/24
# vtysh -c 'show ip bgp version 8 json' | jq -r '.routes | keys[]'
192.168.2.0/24
192.168.3.0/24
```

show bgp [afi] [safi] statistics
Display statistics of routes of the selected afi and safi.

show bgp statistics-all
Display statistics of routes of all the afi and safi.

show [ip] bgp [afi] [safi] [all] cidr-only [wide|json]
Display routes with non-natural netmasks.

show [ip] bgp [afi] [safi] [all] prefix-list WORD [wide|json]
Display routes that match the specified prefix-list.

If `wide` option is specified, then the prefix table's width is increased to fully display the prefix and the nexthop.

If the `json` option is specified, output is displayed in JSON format.

show [ip] bgp [afi] [safi] [all] access-list WORD [wide|json]
Display routes that match the specified access-list.

show [ip] bgp [afi] [safi] [all] filter-list WORD [wide|json]
Display routes that match the specified AS-Path filter-list.

If `wide` option is specified, then the prefix table's width is increased to fully display the prefix and the nexthop.

If the `json` option is specified, output is displayed in JSON format.

show [ip] bgp [afi] [safi] [all] route-map WORD [wide|json]
Display routes that match the specified route-map.

If `wide` option is specified, then the prefix table's width is increased to fully display the prefix and the nexthop.

If the `json` option is specified, output is displayed in JSON format.

show [ip] bgp [afi] [safi] [all] <A.B.C.D/M|X:X::X:X/M> longer-prefixes [wide|json]
Displays the specified route and all more specific routes.

If `wide` option is specified, then the prefix table's width is increased to fully display the prefix and the nexthop.

If the `json` option is specified, output is displayed in JSON format.

show [ip] bgp [afi] [safi] [all] self-originate [wide|json]
Display self-originated routes.

If `wide` option is specified, then the prefix table's width is increased to fully display the prefix and the nexthop.

If the `json` option is specified, output is displayed in JSON format.

show [ip] bgp [afi] [safi] [all] neighbors A.B.C.D [advertised-routes|received-routes|filtered-routes] [<A.B.C.D/M|X:X::X:X/M> | detail] [json|wide]

Display the routes advertised to a BGP neighbor or received routes from neighbor or filtered routes received from neighbor based on the option specified.

If `wide` option is specified, then the prefix table's width is increased to fully display the prefix and the nexthop.

This is especially handy dealing with IPv6 prefixes and if `[no] bgp default show-nexthop-hostname` is enabled.

If `all` option is specified, `ip` keyword is ignored and, routes displayed for all AFIs and SAFIs. if afi is specified, with `all` option, routes will be displayed for each SAFI in the selcted AFI

If a specific prefix is specified, the detailed version of that prefix will be displayed.

If `detail` option is specified, the detailed version of all routes will be displayed. The same format as `show [ip] bgp [afi] [safi] PREFIX` will be used, but for the whole table of received, advertised or filtered prefixes.

If `json` option is specified, output is displayed in JSON format.

show [ip] bgp [afi] [safi] [all] detail-routes

Display the detailed version of all routes. The same format as using `show [ip] bgp [afi] [safi] PREFIX`, but for the whole BGP table.

If `all` option is specified, `ip` keyword is ignored and, routes displayed for all AFIs and SAFIs.

If `afi` is specified, with `all` option, routes will be displayed for each SAFI in the selected AFI.

show [ip] bgp [<view|vrf> VIEWVRFNAME] [afi] [safi] detail [json]

Display the detailed version of all routes from the specified bgp vrf table for a given afi + safi.

If no vrf is specified, then it is assumed as a default vrf and routes are displayed from default vrf table.

If `all` option is specified as vrf name, then all bgp vrf tables routes from a given afi+safi are displayed in the detailed output of routes.

If `json` option is specified, detailed output is displayed in JSON format.

Following are sample output for few examples of how to use this command.

```
torm-23# sh bgp ipv4 unicast detail (OR) sh bgp vrf default ipv4 unicast detail

!--- Output suppressed.

BGP routing table entry for 172.16.16.1/32
Paths: (1 available, best #1, table default)
  Not advertised to any peer
  Local, (Received from a RR-client)
    172.16.16.1 (metric 20) from torm-22(172.16.16.1) (192.168.0.10)
      Origin IGP, metric 0, localpref 100, valid, internal
      Last update: Fri May  8 12:54:05 2023
BGP routing table entry for 172.16.16.2/32
Paths: (1 available, best #1, table default)
  Not advertised to any peer
  Local
    0.0.0.0 from 0.0.0.0 (172.16.16.2)
      Origin incomplete, metric 0, weight 32768, valid, sourced, bestpath-from-AS Local,␣
↪best (First path received)
      Last update: Wed May  8 12:54:41 2023
```

(continues on next page)

(continued from previous page)

```
Displayed  2 routes and 2 total paths
```

```
torm-23# sh bgp vrf all detail

Instance default:

!--- Output suppressed.

BGP routing table entry for 172.16.16.1/32
Paths: (1 available, best #1, table default)
  Not advertised to any peer
  Local, (Received from a RR-client)
    172.16.16.1 (metric 20) from torm-22(172.16.16.1) (192.168.0.10)
      Origin IGP, metric 0, localpref 100, valid, internal
      Last update: Fri May  8 12:44:05 2023
BGP routing table entry for 172.16.16.2/32
Paths: (1 available, best #1, table default)
  Not advertised to any peer
  Local
    0.0.0.0 from 0.0.0.0 (172.16.16.2)
      Origin incomplete, metric 0, weight 32768, valid, sourced, bestpath-from-AS Local,␣
↪best (First path received)
      Last update: Wed May  8 12:45:01 2023

Displayed  2 routes and 2 total paths

Instance vrf3:

!--- Output suppressed.

BGP routing table entry for 192.168.0.2/32
Paths: (1 available, best #1, vrf vrf3)
  Not advertised to any peer
  Imported from 172.16.16.1:12:[2]:[0]:[48]:[00:02:00:00:00:58]:[32]:[192.168.0.2], VNI␣
↪1008/4003
  Local
    172.16.16.1 from torm-22(172.16.16.1) (172.16.16.1) announce-nh-self
      Origin IGP, localpref 100, valid, internal, bestpath-from-AS Local, best (First␣
↪path received)
      Extended Community: RT:65000:1008 ET:8 Rmac:00:02:00:00:00:58
      Last update: Fri May  8 02:41:55 2023
BGP routing table entry for 192.168.1.2/32
Paths: (1 available, best #1, vrf vrf3)
  Not advertised to any peer
  Imported from 172.16.16.1:13:[2]:[0]:[48]:[00:02:00:00:00:58]:[32]:[192.168.1.2], VNI␣
↪1009/4003
  Local
    172.16.16.1 from torm-22(172.16.16.1) (172.16.16.1) announce-nh-self
      Origin IGP, localpref 100, valid, internal, bestpath-from-AS Local, best (First␣
↪path received)
      Extended Community: RT:65000:1009 ET:8 Rmac:00:02:00:00:00:58
```

(continues on next page)

(continued from previous page)

```
      Last update: Fri May  8 02:41:55 2023

Displayed  2 routes and 2 total paths
```

```
torm-23# sh bgp vrf vrf3 ipv4 unicast detail

!--- Output suppressed.

BGP routing table entry for 192.168.0.2/32
Paths: (1 available, best #1, vrf vrf3)
  Not advertised to any peer
  Imported from 172.16.16.1:12:[2]:[0]:[48]:[00:02:00:00:00:58]:[32]:[192.168.0.2], VNI␣
↪1008/4003
  Local
    172.16.16.1 from torm-22(172.16.16.1) (172.16.16.1) announce-nh-self
      Origin IGP, localpref 100, valid, internal, bestpath-from-AS Local, best (First␣
↪path received)
      Extended Community: RT:65000:1008 ET:8 Rmac:00:02:00:00:00:58
      Last update: Fri May  8 02:23:35 2023
BGP routing table entry for 192.168.1.2/32
Paths: (1 available, best #1, vrf vrf3)
  Not advertised to any peer
  Imported from 172.16.16.1:13:[2]:[0]:[48]:[00:02:00:00:00:58]:[32]:[192.168.1.2], VNI␣
↪1009/4003
  Local
    172.16.16.1 from torm-22(172.16.16.1) (172.16.16.1) announce-nh-self
      Origin IGP, localpref 100, valid, internal, bestpath-from-AS Local, best (First␣
↪path received)
      Extended Community: RT:65000:1009 ET:8 Rmac:00:02:00:00:00:58
      Last update: Fri May  8 02:23:55 2023

Displayed  2 routes and 2 total paths
```

Displaying Routes by Community Attribute

The following commands allow displaying routes based on their community attribute.

`show [ip] bgp <ipv4|ipv6> [all] community [wide|json]`

`show [ip] bgp <ipv4|ipv6> [all] community COMMUNITY [wide|json]`

`show [ip] bgp <ipv4|ipv6> [all] community COMMUNITY exact-match [wide|json]`

These commands display BGP routes which have the community attribute. attribute. When `COMMUNITY` is specified, BGP routes that match that community are displayed. When *exact-match* is specified, it display only routes that have an exact match.

`show [ip] bgp <ipv4|ipv6> community-list WORD [json]`

`show [ip] bgp <ipv4|ipv6> community-list WORD exact-match [json]`

These commands display BGP routes for the address family specified that match the specified community list. When *exact-match* is specified, it displays only routes that have an exact match.

If `wide` option is specified, then the prefix table's width is increased to fully display the prefix and the nexthop.

This is especially handy dealing with IPv6 prefixes and if `[no] bgp default show-nexthop-hostname` is enabled.

If `all` option is specified, `ip` keyword is ignored and, routes displayed for all AFIs and SAFIs. if afi is specified, with `all` option, routes will be displayed for each SAFI in the selcted AFI

If `json` option is specified, output is displayed in JSON format.

`show bgp labelpool <chunks|inuse|ledger|requests|summary> [json]`

These commands display information about the BGP labelpool used for the association of MPLS labels with routes for L3VPN and Labeled Unicast

If `chunks` option is specified, output shows the current list of label chunks granted to BGP by Zebra, indicating the start and end label in each chunk

If `inuse` option is specified, output shows the current inuse list of label to prefix mappings

If `ledger` option is specified, output shows ledger list of all label requests made per prefix

If `requests` option is specified, output shows current list of label requests which have not yet been fulfilled by the labelpool

If `summary` option is specified, output is a summary of the counts for the chunks, inuse, ledger and requests list along with the count of outstanding chunk requests to Zebra and the number of zebra reconnects that have happened

If `json` option is specified, output is displayed in JSON format.

Displaying Routes by Large Community Attribute

The following commands allow displaying routes based on their large community attribute.

`show [ip] bgp <ipv4|ipv6> large-community`

`show [ip] bgp <ipv4|ipv6> large-community LARGE-COMMUNITY`

`show [ip] bgp <ipv4|ipv6> large-community LARGE-COMMUNITY exact-match`

`show [ip] bgp <ipv4|ipv6> large-community LARGE-COMMUNITY json`

These commands display BGP routes which have the large community attribute. attribute. When `LARGE-COMMUNITY` is specified, BGP routes that match that large community are displayed. When *exact-match* is specified, it display only routes that have an exact match. When *json* is specified, it display routes in json format.

`show [ip] bgp <ipv4|ipv6> large-community-list WORD`

`show [ip] bgp <ipv4|ipv6> large-community-list WORD exact-match`

`show [ip] bgp <ipv4|ipv6> large-community-list WORD json`

These commands display BGP routes for the address family specified that match the specified large community list. When *exact-match* is specified, it displays only routes that have an exact match. When *json* is specified, it display routes in json format.

Displaying Routes by AS Path

`show bgp ipv4|ipv6 regexp LINE`
: This commands displays BGP routes that matches a regular expression *line* (*BGP Regular Expressions*).

`show [ip] bgp ipv4 vpn`

`show [ip] bgp ipv6 vpn`
: Print active IPV4 or IPV6 routes advertised via the VPN SAFI.

`show bgp ipv4 vpn summary`

`show bgp ipv6 vpn summary`
: Print a summary of neighbor connections for the specified AFI/SAFI combination.

Displaying Routes by Route Distinguisher

`show bgp [<ipv4|ipv6> vpn | l2vpn evpn [route]] rd <all|RD>`
: For L3VPN and EVPN address-families, routes can be displayed on a per-RD (Route Distinguisher) basis or for all RD's.

`show bgp l2vpn evpn rd <all|RD> [overlay | tags]`
: Use the `overlay` or `tags` keywords to display the overlay/tag information about the EVPN prefixes in the selected Route Distinguisher.

`show bgp l2vpn evpn route rd <all|RD> mac <MAC> [ip <MAC>] [json]`
: For EVPN Type 2 (macip) routes, a MAC address (and optionally an IP address) can be supplied to the command to only display matching prefixes in the specified RD.

Displaying Update Group Information

`show bgp update-groups [advertise-queue|advertised-routes|packet-queue]`
: Display Information about each individual update-group being used. If SUBGROUP-ID is specified only display about that particular group. If advertise-queue is specified the list of routes that need to be sent to the peers in the update-group is displayed, advertised-routes means the list of routes we have sent to the peers in the update-group and packet-queue specifies the list of packets in the queue to be sent.

`show bgp update-groups statistics`
: Display Information about update-group events in FRR.

Displaying Nexthop Information

`show [ip] bgp [<view|vrf> VIEWVRFNAME] nexthop ipv4 [A.B.C.D] [detail] [json]`

`show [ip] bgp [<view|vrf> VIEWVRFNAME] nexthop ipv6 [X:X::X:X] [detail] [json]`

`show [ip] bgp [<view|vrf> VIEWVRFNAME] nexthop [<A.B.C.D|X:X::X:X>] [detail] [json]`

`show [ip] bgp <view|vrf> all nexthop [json]`
: Display information about nexthops to bgp neighbors. If a certain nexthop is specified, also provides information about paths associated with the nexthop. With detail option provides information about gates of each nexthop.

`show [ip] bgp [<view|vrf> VIEWVRFNAME] import-check-table [detail] [json]`
: Display information about nexthops from table that is used to check network's existence in the rib for network statements.

Segment-Routing IPv6

show bgp segment-routing srv6

This command displays information about SRv6 L3VPN in bgpd. Specifically, what kind of Locator is being used, and its Locator chunk information. And the SID of the SRv6 Function that is actually managed on bgpd. In the following example, bgpd is using a Locator named loc1, and two SRv6 Functions are managed to perform VPNv6 VRF redirect for vrf10 and vrf20.

```
router# show bgp segment-routing srv6
locator_name: loc1
locator_chunks:
- 2001:db8:1:1::/64
functions:
- sid: 2001:db8:1:1::100
  locator: loc1
- sid: 2001:db8:1:1::200
  locator: loc1
bgps:
- name: default
  vpn_policy[AFI_IP].tovpn_sid: none
  vpn_policy[AFI_IP6].tovpn_sid: none
- name: vrf10
  vpn_policy[AFI_IP].tovpn_sid: none
  vpn_policy[AFI_IP6].tovpn_sid: 2001:db8:1:1::100
- name: vrf20
  vpn_policy[AFI_IP].tovpn_sid: none
  vpn_policy[AFI_IP6].tovpn_sid: 2001:db8:1:1::200
```

AS-notation support

By default, the ASN value output follows how the BGP ASN instance is expressed in the configuration. Three as-notation outputs are available:

- plain output: both AS4B and AS2B use a single number. ` router bgp 65536`.
- dot output: AS4B values are using two numbers separated by a period. *router bgp 1.1* means that the AS number is 65536.
- dot+ output: AS2B and AS4B values are using two numbers separated by a period. *router bgp 0.5* means that the AS number is 5.

The below option permits forcing the as-notation output:

router bgp ASN as-notation dot|dot+|plain

The chosen as-notation format will override the BGP ASN output.

3.3.5 Route Reflector

BGP routers connected inside the same AS through BGP belong to an internal BGP session, or IBGP. In order to prevent routing table loops, IBGP does not advertise IBGP-learned routes to other routers in the same session. As such, IBGP requires a full mesh of all peers. For large networks, this quickly becomes unscalable. Introducing route reflectors removes the need for the full-mesh.

When route reflectors are configured, these will reflect the routes announced by the peers configured as clients. A route reflector client is configured with:

`neighbor PEER route-reflector-client`

To avoid single points of failure, multiple route reflectors can be configured.

A cluster is a collection of route reflectors and their clients, and is used by route reflectors to avoid looping.

`bgp cluster-id A.B.C.D`

`bgp no-rib`

To set and unset the BGP daemon `-n` / `--no_kernel` options during runtime to disable BGP route installation to the RIB (Zebra), the `[no] bgp no-rib` commands can be used;

Please note that setting the option during runtime will withdraw all routes in the daemons RIB from Zebra and unsetting it will announce all routes in the daemons RIB to Zebra. If the option is passed as a command line argument when starting the daemon and the configuration gets saved, the option will persist unless removed from the configuration with the negating command prior to the configuration write operation. At this point in time non SAFI_UNICAST BGP data is not properly withdrawn from zebra when this command is issued.

`bgp allow-martian-nexthop`

When a peer receives a martian nexthop as part of the NLRI for a route permit the nexthop to be used as such, instead of rejecting and resetting the connection.

`bgp send-extra-data zebra`

This command turns on the ability of BGP to send extra data to zebra. Currently, it's the AS-Path, communities, and the path selection reason. The default behavior in BGP is not to send this data. If the routes were sent to zebra and the option is changed, bgpd doesn't reinstall the routes to comply with the new setting.

`bgp session-dscp (0-63)`

This command allows bgp to control, at a global level, the TCP dscp values in the TCP header.

3.3.6 Suppressing routes not installed in FIB

The FRR implementation of BGP advertises prefixes learnt from a peer to other peers even if the routes do not get installed in the FIB. There can be scenarios where the hardware tables in some of the routers (along the path from the source to destination) is full which will result in all routes not getting installed in the FIB. If these routes are advertised to the downstream routers then traffic will start flowing and will be dropped at the intermediate router.

The solution is to provide a configurable option to check for the FIB install status of the prefixes and advertise to peers if the prefixes are successfully installed in the FIB. The advertisement of the prefixes are suppressed if it is not installed in FIB.

The following conditions apply will apply when checking for route installation status in FIB:

1. The advertisement or suppression of routes based on FIB install status applies only for newly learnt routes from peer (routes which are not in BGP local RIB).

2. If the route received from peer already exists in BGP local RIB and route attributes have changed (best path changed), the old path is deleted and new path is installed in FIB. The FIB install status will not have any effect. Therefore only when the route is received first time the checks apply.
3. The feature will not apply for routes learnt through other means like redistribution to bgp from other protocols. This is applicable only to peer learnt routes.
4. If a route is installed in FIB and then gets deleted from the dataplane, then routes will not be withdrawn from peers. This will be considered as dataplane issue.
5. The feature will slightly increase the time required to advertise the routes to peers since the route install status needs to be received from the FIB
6. If routes are received by the peer before the configuration is applied, then the bgp sessions need to be reset for the configuration to take effect.
7. If the route which is already installed in dataplane is removed for some reason, sending withdraw message to peers is not currently supported.

`bgp suppress-fib-pending`
This command is applicable at the global level and at an individual bgp level. If applied at the global level all bgp instances will wait for fib installation before announcing routes and there is no way to turn it off for a particular bgp vrf.

3.3.7 Routing Policy

You can set different routing policy for a peer. For example, you can set different filter for a peer.

```
!
router bgp 1 view 1
 neighbor 10.0.0.1 remote-as 2
 address-family ipv4 unicast
  neighbor 10.0.0.1 distribute-list 1 in
 exit-address-family
!
router bgp 1 view 2
 neighbor 10.0.0.1 remote-as 2
 address-family ipv4 unicast
  neighbor 10.0.0.1 distribute-list 2 in
 exit-address-family
```

This means BGP update from a peer 10.0.0.1 goes to both BGP view 1 and view 2. When the update is inserted into view 1, distribute-list 1 is applied. On the other hand, when the update is inserted into view 2, distribute-list 2 is applied.

3.3.8 BGP Regular Expressions

BGP regular expressions are based on *POSIX 1003.2* regular expressions. The following description is just a quick subset of the POSIX regular expressions.

.* Matches any single character.

***** Matches 0 or more occurrences of pattern.

+ Matches 1 or more occurrences of pattern.

? Match 0 or 1 occurrences of pattern.

^ Matches the beginning of the line.

$ Matches the end of the line.

_ The _ character has special meanings in BGP regular expressions. It matches to space and comma , and AS set delimiter { and } and AS confederation delimiter (and). And it also matches to the beginning of the line and the end of the line. So _ can be used for AS value boundaries match. This character technically evaluates to `(^|[,{}()]|$)`.

3.3.9 Miscellaneous Configuration Examples

Example of a session to an upstream, advertising only one prefix to it.

```
router bgp 64512
 bgp router-id 10.236.87.1
 neighbor upstream peer-group
 neighbor upstream remote-as 64515
 neighbor upstream capability dynamic
 neighbor 10.1.1.1 peer-group upstream
 neighbor 10.1.1.1 description ACME ISP

 address-family ipv4 unicast
  network 10.236.87.0/24
  neighbor upstream prefix-list pl-allowed-adv out
 exit-address-family
!
ip prefix-list pl-allowed-adv seq 5 permit 82.195.133.0/25
ip prefix-list pl-allowed-adv seq 10 deny any
```

A more complex example including upstream, peer and customer sessions advertising global prefixes and NO_EXPORT prefixes and providing actions for customer routes based on community values. Extensive use is made of route-maps and the ‘call’ feature to support selective advertising of prefixes. This example is intended as guidance only, it has NOT been tested and almost certainly contains silly mistakes, if not serious flaws.

```
router bgp 64512
 bgp router-id 10.236.87.1
 neighbor upstream capability dynamic
 neighbor cust capability dynamic
 neighbor peer capability dynamic
 neighbor 10.1.1.1 remote-as 64515
 neighbor 10.1.1.1 peer-group upstream
 neighbor 10.2.1.1 remote-as 64516
 neighbor 10.2.1.1 peer-group upstream
 neighbor 10.3.1.1 remote-as 64517
 neighbor 10.3.1.1 peer-group cust-default
 neighbor 10.3.1.1 description customer1
 neighbor 10.4.1.1 remote-as 64518
 neighbor 10.4.1.1 peer-group cust
 neighbor 10.4.1.1 description customer2
 neighbor 10.5.1.1 remote-as 64519
 neighbor 10.5.1.1 peer-group peer
 neighbor 10.5.1.1 description peer AS 1
 neighbor 10.6.1.1 remote-as 64520
 neighbor 10.6.1.1 peer-group peer
 neighbor 10.6.1.1 description peer AS 2
```

(continues on next page)

(continued from previous page)

```
 address-family ipv4 unicast
  network 10.123.456.0/24
  network 10.123.456.128/25 route-map rm-no-export
  neighbor upstream route-map rm-upstream-out out
  neighbor cust route-map rm-cust-in in
  neighbor cust route-map rm-cust-out out
  neighbor cust send-community both
  neighbor peer route-map rm-peer-in in
  neighbor peer route-map rm-peer-out out
  neighbor peer send-community both
  neighbor 10.3.1.1 prefix-list pl-cust1-network in
  neighbor 10.4.1.1 prefix-list pl-cust2-network in
  neighbor 10.5.1.1 prefix-list pl-peer1-network in
  neighbor 10.6.1.1 prefix-list pl-peer2-network in
 exit-address-family
!
ip prefix-list pl-default permit 0.0.0.0/0
!
ip prefix-list pl-upstream-peers permit 10.1.1.1/32
ip prefix-list pl-upstream-peers permit 10.2.1.1/32
!
ip prefix-list pl-cust1-network permit 10.3.1.0/24
ip prefix-list pl-cust1-network permit 10.3.2.0/24
!
ip prefix-list pl-cust2-network permit 10.4.1.0/24
!
ip prefix-list pl-peer1-network permit 10.5.1.0/24
ip prefix-list pl-peer1-network permit 10.5.2.0/24
ip prefix-list pl-peer1-network permit 192.168.0.0/24
!
ip prefix-list pl-peer2-network permit 10.6.1.0/24
ip prefix-list pl-peer2-network permit 10.6.2.0/24
ip prefix-list pl-peer2-network permit 192.168.1.0/24
ip prefix-list pl-peer2-network permit 192.168.2.0/24
ip prefix-list pl-peer2-network permit 172.16.1/24
!
bgp as-path access-list seq 5 asp-own-as permit ^$
bgp as-path access-list seq 10 asp-own-as permit _64512_
!
! ################################################################
! Match communities we provide actions for, on routes receives from
! customers. Communities values of <our-ASN>:X, with X, have actions:
!
! 100 - blackhole the prefix
! 200 - set no_export
! 300 - advertise only to other customers
! 400 - advertise only to upstreams
! 500 - set no_export when advertising to upstreams
! 2X00 - set local_preference to X00
!
! blackhole the prefix of the route
```

(continues on next page)

(continued from previous page)

```
bgp community-list standard cm-blackhole permit 64512:100
!
! set no-export community before advertising
bgp community-list standard cm-set-no-export permit 64512:200
!
! advertise only to other customers
bgp community-list standard cm-cust-only permit 64512:300
!
! advertise only to upstreams
bgp community-list standard cm-upstream-only permit 64512:400
!
! advertise to upstreams with no-export
bgp community-list standard cm-upstream-noexport permit 64512:500
!
! set local-pref to least significant 3 digits of the community
bgp community-list standard cm-prefmod-100 permit 64512:2100
bgp community-list standard cm-prefmod-200 permit 64512:2200
bgp community-list standard cm-prefmod-300 permit 64512:2300
bgp community-list standard cm-prefmod-400 permit 64512:2400
bgp community-list expanded cme-prefmod-range permit 64512:2...
!
! Informational communities
!
! 3000 - learned from upstream
! 3100 - learned from customer
! 3200 - learned from peer
!
bgp community-list standard cm-learnt-upstream permit 64512:3000
bgp community-list standard cm-learnt-cust permit 64512:3100
bgp community-list standard cm-learnt-peer permit 64512:3200
!
! ###################################################################
! Utility route-maps
!
! These utility route-maps generally should not used to permit/deny
! routes, i.e. they do not have meaning as filters, and hence probably
! should be used with 'on-match next'. These all finish with an empty
! permit entry so as not interfere with processing in the caller.
!
route-map rm-no-export permit 10
 set community additive no-export
route-map rm-no-export permit 20
!
route-map rm-blackhole permit 10
 description blackhole, up-pref and ensure it cannot escape this AS
 set ip next-hop 127.0.0.1
 set local-preference 10
 set community additive no-export
route-map rm-blackhole permit 20
!
! Set local-pref as requested
route-map rm-prefmod permit 10
```

(continues on next page)

(continued from previous page)

```
 match community cm-prefmod-100
 set local-preference 100
route-map rm-prefmod permit 20
 match community cm-prefmod-200
 set local-preference 200
route-map rm-prefmod permit 30
 match community cm-prefmod-300
 set local-preference 300
route-map rm-prefmod permit 40
 match community cm-prefmod-400
 set local-preference 400
route-map rm-prefmod permit 50
!
! Community actions to take on receipt of route.
route-map rm-community-in permit 10
 description check for blackholing, no point continuing if it matches.
 match community cm-blackhole
 call rm-blackhole
route-map rm-community-in permit 20
 match community cm-set-no-export
 call rm-no-export
 on-match next
route-map rm-community-in permit 30
 match community cme-prefmod-range
 call rm-prefmod
route-map rm-community-in permit 40
!
! ###################################################################
! Community actions to take when advertising a route.
! These are filtering route-maps,
!
! Deny customer routes to upstream with cust-only set.
route-map rm-community-filt-to-upstream deny 10
 match community cm-learnt-cust
 match community cm-cust-only
route-map rm-community-filt-to-upstream permit 20
!
! Deny customer routes to other customers with upstream-only set.
route-map rm-community-filt-to-cust deny 10
 match community cm-learnt-cust
 match community cm-upstream-only
route-map rm-community-filt-to-cust permit 20
!
! ###################################################################
! The top-level route-maps applied to sessions. Further entries could
! be added obviously..
!
! Customers
route-map rm-cust-in permit 10
 call rm-community-in
 on-match next
route-map rm-cust-in permit 20
```

(continues on next page)

(continued from previous page)

```
 set community additive 64512:3100
route-map rm-cust-in permit 30
!
route-map rm-cust-out permit 10
 call rm-community-filt-to-cust
 on-match next
route-map rm-cust-out permit 20
!
! Upstream transit ASes
route-map rm-upstream-out permit 10
 description filter customer prefixes which are marked cust-only
 call rm-community-filt-to-upstream
 on-match next
route-map rm-upstream-out permit 20
 description only customer routes are provided to upstreams/peers
 match community cm-learnt-cust
!
! Peer ASes
! outbound policy is same as for upstream
route-map rm-peer-out permit 10
 call rm-upstream-out
!
route-map rm-peer-in permit 10
 set community additive 64512:3200
```

Example of how to set up a 6-Bone connection.

```
! bgpd configuration
! ==================
!
! MP-BGP configuration
!
router bgp 7675
 bgp router-id 10.0.0.1
 neighbor 3ffe:1cfa:0:2:2a0:c9ff:fe9e:f56 remote-as `as-number`
!
 address-family ipv6
 network 3ffe:506::/32
 neighbor 3ffe:1cfa:0:2:2a0:c9ff:fe9e:f56 activate
 neighbor 3ffe:1cfa:0:2:2a0:c9ff:fe9e:f56 route-map set-nexthop out
 neighbor 3ffe:1cfa:0:2:2c0:4fff:fe68:a231 remote-as `as-number`
 neighbor 3ffe:1cfa:0:2:2c0:4fff:fe68:a231 route-map set-nexthop out
 exit-address-family
!
ipv6 access-list all permit any
!
! Set output nexthop address.
!
route-map set-nexthop permit 10
 match ipv6 address all
 set ipv6 nexthop global 3ffe:1cfa:0:2:2c0:4fff:fe68:a225
 set ipv6 nexthop local fe80::2c0:4fff:fe68:a225
```

(continues on next page)

(continued from previous page)

```
!
log file bgpd.log
!
```

3.3.10 BGP tcp-mss support

TCP provides a mechanism for the user to specify the max segment size. setsockopt API is used to set the max segment size for TCP session. We can configure this as part of BGP neighbor configuration.

This document explains how to avoid ICMP vulnerability issues by limiting TCP max segment size when you are using MTU discovery. Using MTU discovery on TCP paths is one method of avoiding BGP packet fragmentation.

TCP negotiates a maximum segment size (MSS) value during session connection establishment between two peers. The MSS value negotiated is primarily based on the maximum transmission unit (MTU) of the interfaces to which the communicating peers are directly connected. However, due to variations in link MTU on the path taken by the TCP packets, some packets in the network that are well within the MSS value might be fragmented when the packet size exceeds the link's MTU.

This feature is supported with TCP over IPv4 and TCP over IPv6.

CLI Configuration:

Below configuration can be done in router bgp mode and allows the user to configure the tcp-mss value per neighbor. The configuration gets applied only after hard reset is performed on that neighbor. If we configure tcp-mss on both the neighbors then both neighbors need to be reset.

The configuration takes effect based on below rules, so there is a configured tcp-mss and a synced tcp-mss value per TCP session.

By default if the configuration is not done then the TCP max segment size is set to the Maximum Transmission unit (MTU) – (IP/IP6 header size + TCP header size + ethernet header). For IPv4 its MTU – (20 bytes IP header + 20 bytes TCP header + 12 bytes ethernet header) and for IPv6 its MTU – (40 bytes IPv6 header + 20 bytes TCP header + 12 bytes ethernet header).

If the config is done then it reduces 12-14 bytes for the ether header and uses it after synchronizing in TCP handshake.

`neighbor <A.B.C.D|X:X::X:X|WORD> tcp-mss (1-65535)`

When tcp-mss is configured kernel reduces 12-14 bytes for ethernet header. E.g. if tcp-mss is configured as 150 the synced value will be 138.

Note: configured and synced value is different since TCP module will reduce 12 bytes for ethernet header.

Running config:

```
frr# show running-config
Building configuration...

Current configuration:
!
router bgp 100
 bgp router-id 192.0.2.1
```

(continues on next page)

(continued from previous page)

```
 neighbor 198.51.100.2 remote-as 100
 neighbor 198.51.100.2 tcp-mss 150       => new entry
 neighbor 2001:DB8::2 remote-as 100
 neighbor 2001:DB8::2 tcp-mss 400        => new entry
```

Show command:

```
frr# show bgp neighbors 198.51.100.2
BGP neighbor is 198.51.100.2, remote AS 100, local AS 100, internal link
Hostname: frr
  BGP version 4, remote router ID 192.0.2.2, local router ID 192.0.2.1
  BGP state = Established, up for 02:15:28
  Last read 00:00:28, Last write 00:00:28
  Hold time is 180, keepalive interval is 60 seconds
  Configured tcp-mss is 150, synced tcp-mss is 138     => new display
```

```
frr# show bgp neighbors 2001:DB8::2
BGP neighbor is 2001:DB8::2, remote AS 100, local AS 100, internal link
Hostname: frr
  BGP version 4, remote router ID 192.0.2.2, local router ID 192.0.2.1
  BGP state = Established, up for 02:16:34
  Last read 00:00:34, Last write 00:00:34
  Hold time is 180, keepalive interval is 60 seconds
  Configured tcp-mss is 400, synced tcp-mss is 388      => new display
```

Show command json output:

```
frr# show bgp neighbors 2001:DB8::2 json
{
  "2001:DB8::2":{
    "remoteAs":100,
    "localAs":100,
    "nbrInternalLink":true,
    "hostname":"frr",
    "bgpVersion":4,
    "remoteRouterId":"192.0.2.2",
    "localRouterId":"192.0.2.1",
    "bgpState":"Established",
    "bgpTimerUpMsec":8349000,
    "bgpTimerUpString":"02:19:09",
    "bgpTimerUpEstablishedEpoch":1613054251,
    "bgpTimerLastRead":9000,
    "bgpTimerLastWrite":9000,
    "bgpInUpdateElapsedTimeMsecs":8347000,
    "bgpTimerHoldTimeMsecs":180000,
    "bgpTimerKeepAliveIntervalMsecs":60000,
    "bgpTcpMssConfigured":400,                                   => new entry
    "bgpTcpMssSynced":388,                                  => new entry
```

```
frr# show bgp neighbors 198.51.100.2 json
{
  "198.51.100.2":{
    "remoteAs":100,
    "localAs":100,
    "nbrInternalLink":true,
    "hostname":"frr",
    "bgpVersion":4,
    "remoteRouterId":"192.0.2.2",
    "localRouterId":"192.0.2.1",
    "bgpState":"Established",
    "bgpTimerUpMsec":8370000,
    "bgpTimerUpString":"02:19:30",
    "bgpTimerUpEstablishedEpoch":1613054251,
    "bgpTimerLastRead":30000,
    "bgpTimerLastWrite":30000,
    "bgpInUpdateElapsedTimeMsecs":8368000,
    "bgpTimerHoldTimeMsecs":180000,
    "bgpTimerKeepAliveIntervalMsecs":60000,
    "bgpTcpMssConfigured":150,                                 => new entry
    "bgpTcpMssSynced":138,                                 => new entry
```

3.3.11 Configuring FRR as a Route Server

The purpose of a Route Server is to centralize the peerings between BGP speakers. For example if we have an exchange point scenario with four BGP speakers, each of which maintaining a BGP peering with the other three (*Full Mesh*), we can convert it into a centralized scenario where each of the four establishes a single BGP peering against the Route Server (*Route server and clients*).

We will first describe briefly the Route Server model implemented by FRR. We will explain the commands that have been added for configuring that model. And finally we will show a full example of FRR configured as Route Server.

Description of the Route Server model

First we are going to describe the normal processing that BGP announcements suffer inside a standard BGP speaker, as shown in *Announcement processing inside a 'normal' BGP speaker*, it consists of three steps:

- When an announcement is received from some peer, the *In* filters configured for that peer are applied to the announcement. These filters can reject the announcement, accept it unmodified, or accept it with some of its attributes modified.
- The announcements that pass the *In* filters go into the Best Path Selection process, where they are compared to other announcements referred to the same destination that have been received from different peers (in case such other announcements exist). For each different destination, the announcement which is selected as the best is inserted into the BGP speaker's Loc-RIB.
- The routes which are inserted in the Loc-RIB are considered for announcement to all the peers (except the one from which the route came). This is done by passing the routes in the Loc-RIB through the *Out* filters corresponding to each peer. These filters can reject the route, accept it unmodified, or accept it with some of its attributes modified. Those routes which are accepted by the *Out* filters of a peer are announced to that peer.

Of course we want that the routing tables obtained in each of the routers are the same when using the route server than when not. But as a consequence of having a single BGP peering (against the route server), the BGP speakers can no longer distinguish from/to which peer each announce comes/goes.

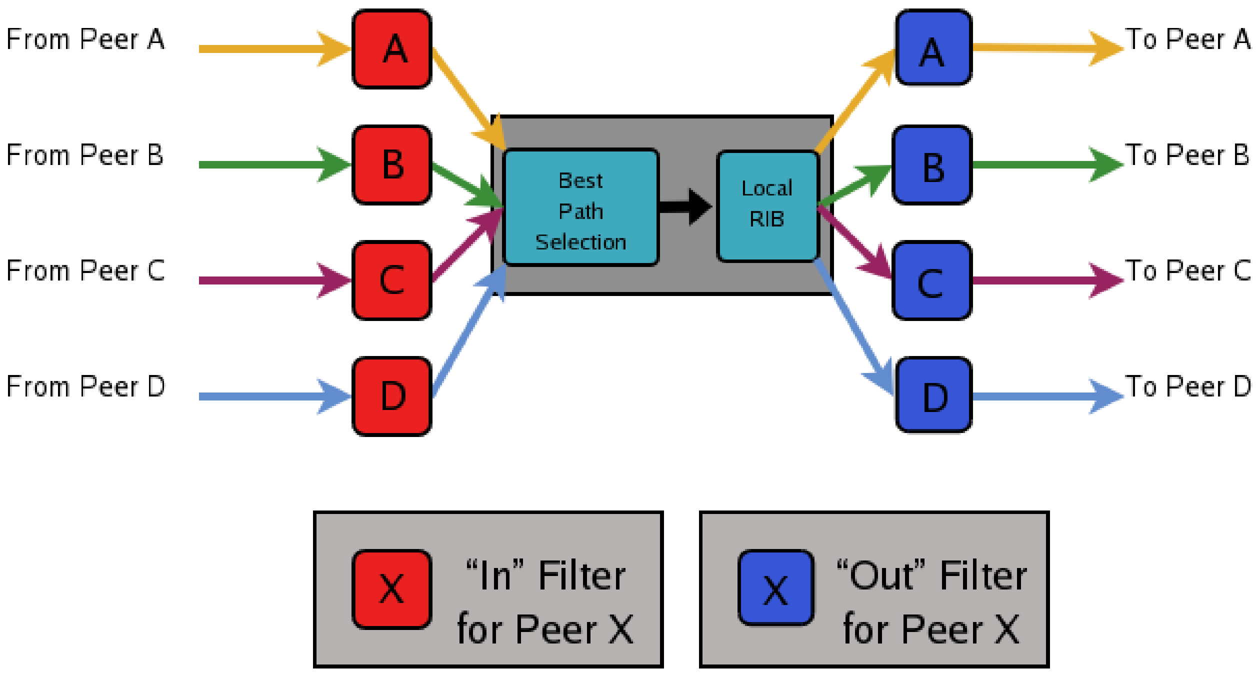

Fig. 1: Announcement processing inside a 'normal' BGP speaker

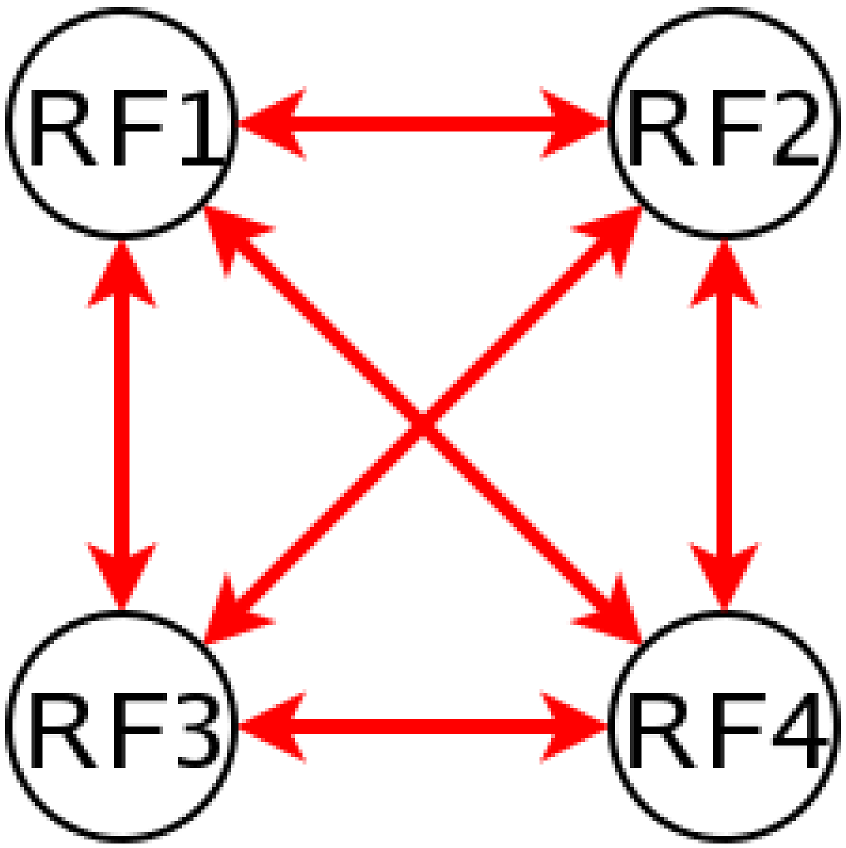

Fig. 2: Full Mesh

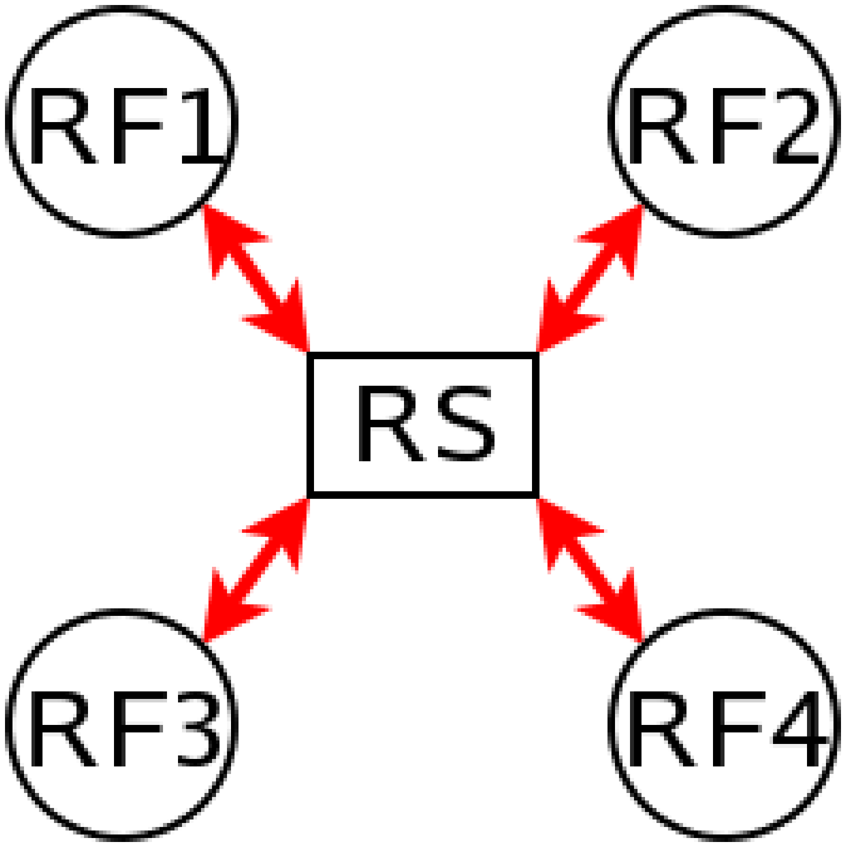

Fig. 3: Route server and clients

This means that the routers connected to the route server are not able to apply by themselves the same input/output filters as in the full mesh scenario, so they have to delegate those functions to the route server.

Even more, the 'best path' selection must be also performed inside the route server on behalf of its clients. The reason is that if, after applying the filters of the announcer and the (potential) receiver, the route server decides to send to some client two or more different announcements referred to the same destination, the client will only retain the last one, considering it as an implicit withdrawal of the previous announcements for the same destination. This is the expected behavior of a BGP speaker as defined in **RFC 1771**, and even though there are some proposals of mechanisms that permit multiple paths for the same destination to be sent through a single BGP peering, none are currently supported by most existing BGP implementations.

As a consequence a route server must maintain additional information and perform additional tasks for a RS-client that those necessary for common BGP peerings. Essentially a route server must:

- Maintain a separated Routing Information Base (Loc-RIB) for each peer configured as RS-client, containing the routes selected as a result of the 'Best Path Selection' process that is performed on behalf of that RS-client.
- Whenever it receives an announcement from a RS-client, it must consider it for the Loc-RIBs of the other RS-clients.
  - This means that for each of them the route server must pass the announcement through the appropriate *Out* filter of the announcer.
  - Then through the appropriate *In* filter of the potential receiver.
  - Only if the announcement is accepted by both filters it will be passed to the 'Best Path Selection' process.
  - Finally, it might go into the Loc-RIB of the receiver.

When we talk about the 'appropriate' filter, both the announcer and the receiver of the route must be taken into account. Suppose that the route server receives an announcement from client A, and the route server is considering it for the Loc-RIB of client B. The filters that should be applied are the same that would be used in the full mesh scenario, i.e., first the *Out* filter of router A for announcements going to router B, and then the *In* filter of router B for announcements coming from router A.

We call 'Export Policy' of a RS-client to the set of *Out* filters that the client would use if there was no route server. The same applies for the 'Import Policy' of a RS-client and the set of *In* filters of the client if there was no route server.

It is also common to demand from a route server that it does not modify some BGP attributes (next-hop, as-path and MED) that are usually modified by standard BGP speakers before announcing a route.

The announcement processing model implemented by FRR is shown in *Announcement processing model implemented by the Route Server*. The figure shows a mixture of RS-clients (B, C and D) with normal BGP peers (A). There are some details that worth additional comments:

- Announcements coming from a normal BGP peer are also considered for the Loc-RIBs of all the RS-clients. But logically they do not pass through any export policy.
- Those peers that are configured as RS-clients do not receive any announce from the *Main* Loc-RIB.
- Apart from import and export policies, *In* and *Out* filters can also be set for RS-clients. *In* filters might be useful when the route server has also normal BGP peers. On the other hand, *Out* filters for RS-clients are probably unnecessary, but we decided not to remove them as they do not hurt anybody (they can always be left empty).

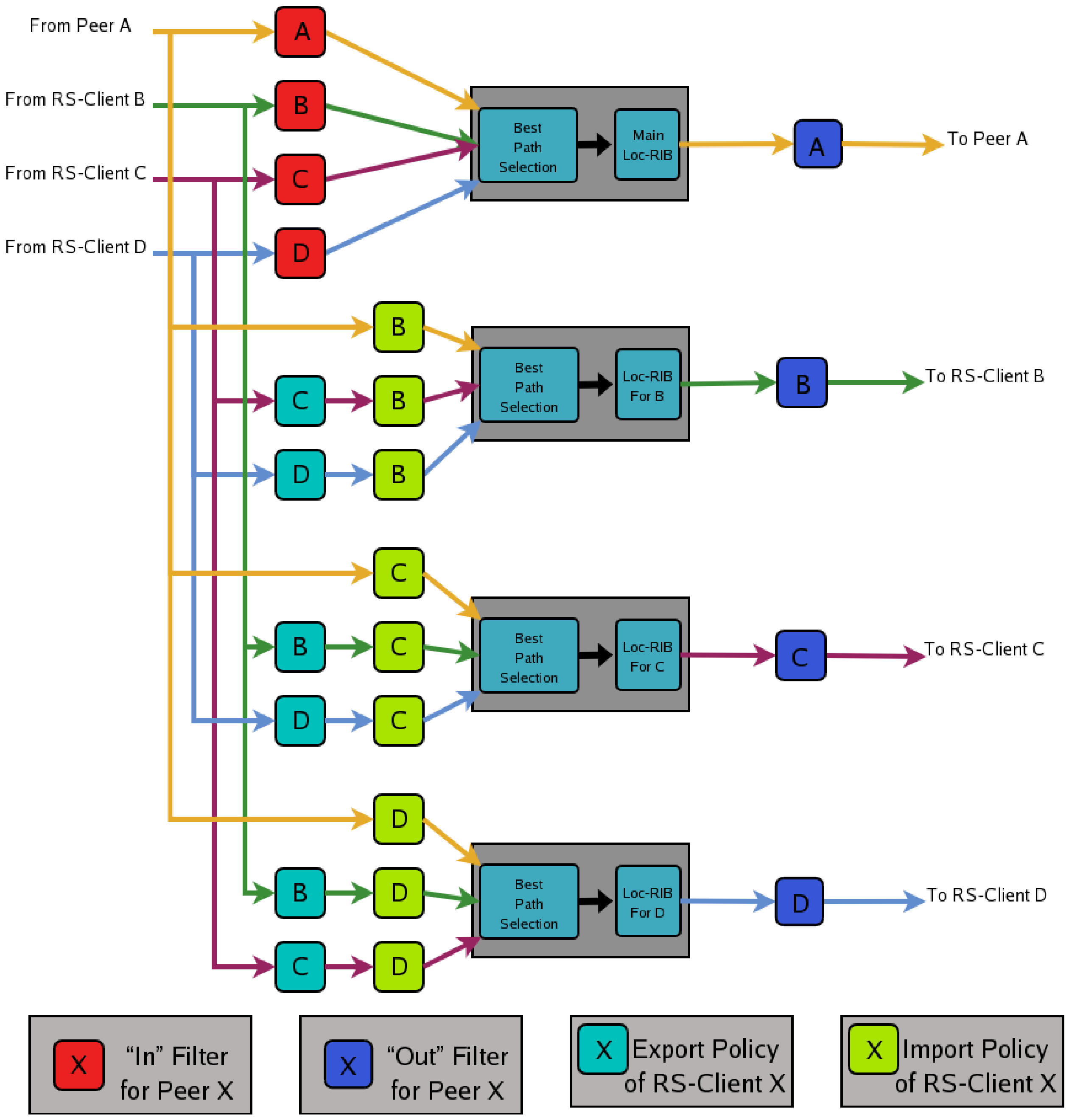

Fig. 4: Announcement processing model implemented by the Route Server

Commands for configuring a Route Server

Now we will describe the commands that have been added to frr in order to support the route server features.

`neighbor PEER-GROUP route-server-client`

`neighbor A.B.C.D route-server-client`

`neighbor X:X::X:X route-server-client`

This command configures the peer given by *peer*, *A.B.C.D* or *X:X::X:X* as an RS-client.

Actually this command is not new, it already existed in standard FRR. It enables the transparent mode for the specified peer. This means that some BGP attributes (as-path, next-hop and MED) of the routes announced to that peer are not modified.

With the route server patch, this command, apart from setting the transparent mode, creates a new Loc-RIB dedicated to the specified peer (those named *Loc-RIB for X* in *Announcement processing model implemented by the Route Server*.). Starting from that moment, every announcement received by the route server will be also considered for the new Loc-RIB.

`neigbor A.B.C.D|X.X::X.X|peer-group route-map WORD in|out`

This set of commands can be used to specify the route-map that represents the Import or Export policy of a peer which is configured as a RS-client (with the previous command).

`match peer A.B.C.D|X:X::X:X`

This is a new *match* statement for use in route-maps, enabling them to describe import/export policies. As we said before, an import/export policy represents a set of input/output filters of the RS-client. This statement makes possible that a single route-map represents the full set of filters that a BGP speaker would use for its different peers in a non-RS scenario.

The *match peer* statement has different semantics whether it is used inside an import or an export route-map. In the first case the statement matches if the address of the peer who sends the announce is the same that the address specified by {A.B.C.D|X:X::X:X}. For export route-maps it matches when {A.B.C.D|X:X::X:X} is the address of the RS-Client into whose Loc-RIB the announce is going to be inserted (how the same export policy is applied before different Loc-RIBs is shown in *Announcement processing model implemented by the Route Server*.).

`call WORD`

This command (also used inside a route-map) jumps into a different route-map, whose name is specified by *WORD*. When the called route-map finishes, depending on its result the original route-map continues or not. Apart from being useful for making import/export route-maps easier to write, this command can also be used inside any normal (in or out) route-map.

Example of Route Server Configuration

Finally we are going to show how to configure a FRR daemon to act as a Route Server. For this purpose we are going to present a scenario without route server, and then we will show how to use the configurations of the BGP routers to generate the configuration of the route server.

All the configuration files shown in this section have been taken from scenarios which were tested using the VNUML tool http://www.dit.upm.es/vnuml,VNUML.

Configuration of the BGP routers without Route Server

We will suppose that our initial scenario is an exchange point with three BGP capable routers, named RA, RB and RC. Each of the BGP speakers generates some routes (with the *network* command), and establishes BGP peerings against the other two routers. These peerings have In and Out route-maps configured, named like 'PEER-X-IN' or 'PEER-X-OUT'. For example the configuration file for router RA could be the following:

```
#Configuration for router 'RA'
!
hostname RA
password ****
!
router bgp 65001
  no bgp default ipv4-unicast
  neighbor 2001:0DB8::B remote-as 65002
  neighbor 2001:0DB8::C remote-as 65003
!
  address-family ipv6
    network 2001:0DB8:AAAA:1::/64
    network 2001:0DB8:AAAA:2::/64
    network 2001:0DB8:0000:1::/64
    network 2001:0DB8:0000:2::/64
    neighbor 2001:0DB8::B activate
    neighbor 2001:0DB8::B soft-reconfiguration inbound
    neighbor 2001:0DB8::B route-map PEER-B-IN in
    neighbor 2001:0DB8::B route-map PEER-B-OUT out
    neighbor 2001:0DB8::C activate
    neighbor 2001:0DB8::C soft-reconfiguration inbound
    neighbor 2001:0DB8::C route-map PEER-C-IN in
    neighbor 2001:0DB8::C route-map PEER-C-OUT out
  exit-address-family
!
ipv6 prefix-list COMMON-PREFIXES seq  5 permit 2001:0DB8:0000::/48 ge 64 le 64
ipv6 prefix-list COMMON-PREFIXES seq 10 deny any
!
ipv6 prefix-list PEER-A-PREFIXES seq  5 permit 2001:0DB8:AAAA::/48 ge 64 le 64
ipv6 prefix-list PEER-A-PREFIXES seq 10 deny any
!
ipv6 prefix-list PEER-B-PREFIXES seq  5 permit 2001:0DB8:BBBB::/48 ge 64 le 64
ipv6 prefix-list PEER-B-PREFIXES seq 10 deny any
!
ipv6 prefix-list PEER-C-PREFIXES seq  5 permit 2001:0DB8:CCCC::/48 ge 64 le 64
ipv6 prefix-list PEER-C-PREFIXES seq 10 deny any
!
route-map PEER-B-IN permit 10
  match ipv6 address prefix-list COMMON-PREFIXES
  set metric 100
route-map PEER-B-IN permit 20
  match ipv6 address prefix-list PEER-B-PREFIXES
  set community 65001:11111
!
route-map PEER-C-IN permit 10
  match ipv6 address prefix-list COMMON-PREFIXES
```

(continues on next page)

(continued from previous page)

```
  set metric 200
route-map PEER-C-IN permit 20
  match ipv6 address prefix-list PEER-C-PREFIXES
  set community 65001:22222
!
route-map PEER-B-OUT permit 10
  match ipv6 address prefix-list PEER-A-PREFIXES
!
route-map PEER-C-OUT permit 10
  match ipv6 address prefix-list PEER-A-PREFIXES
!
line vty
!
```

Configuration of the BGP routers with Route Server

To convert the initial scenario into one with route server, first we must modify the configuration of routers RA, RB and RC. Now they must not peer between them, but only with the route server. For example, RA's configuration would turn into:

```
# Configuration for router 'RA'
!
hostname RA
password ****
!
router bgp 65001
  no bgp default ipv4-unicast
  neighbor 2001:0DB8::FFFF remote-as 65000
!
  address-family ipv6
    network 2001:0DB8:AAAA:1::/64
    network 2001:0DB8:AAAA:2::/64
    network 2001:0DB8:0000:1::/64
    network 2001:0DB8:0000:2::/64

    neighbor 2001:0DB8::FFFF activate
    neighbor 2001:0DB8::FFFF soft-reconfiguration inbound
  exit-address-family
!
line vty
!
```

Which is logically much simpler than its initial configuration, as it now maintains only one BGP peering and all the filters (route-maps) have disappeared.

Configuration of the Route Server itself

As we said when we described the functions of a route server (*Description of the Route Server model*), it is in charge of all the route filtering. To achieve that, the In and Out filters from the RA, RB and RC configurations must be converted into Import and Export policies in the route server.

This is a fragment of the route server configuration (we only show the policies for client RA):

```
# Configuration for Route Server ('RS')
!
hostname RS
password ix
!
router bgp 65000 view RS
  no bgp default ipv4-unicast
  neighbor 2001:0DB8::A  remote-as 65001
  neighbor 2001:0DB8::B  remote-as 65002
  neighbor 2001:0DB8::C  remote-as 65003
!
  address-family ipv6
    neighbor 2001:0DB8::A activate
    neighbor 2001:0DB8::A route-server-client
    neighbor 2001:0DB8::A route-map RSCLIENT-A-IMPORT in
    neighbor 2001:0DB8::A route-map RSCLIENT-A-EXPORT out
    neighbor 2001:0DB8::A soft-reconfiguration inbound

    neighbor 2001:0DB8::B activate
    neighbor 2001:0DB8::B route-server-client
    neighbor 2001:0DB8::B route-map RSCLIENT-B-IMPORT in
    neighbor 2001:0DB8::B route-map RSCLIENT-B-EXPORT out
    neighbor 2001:0DB8::B soft-reconfiguration inbound

    neighbor 2001:0DB8::C activate
    neighbor 2001:0DB8::C route-server-client
    neighbor 2001:0DB8::C route-map RSCLIENT-C-IMPORT in
    neighbor 2001:0DB8::C route-map RSCLIENT-C-EXPORT out
    neighbor 2001:0DB8::C soft-reconfiguration inbound
  exit-address-family
!
ipv6 prefix-list COMMON-PREFIXES seq  5 permit 2001:0DB8:0000::/48 ge 64 le 64
ipv6 prefix-list COMMON-PREFIXES seq 10 deny any
!
ipv6 prefix-list PEER-A-PREFIXES seq  5 permit 2001:0DB8:AAAA::/48 ge 64 le 64
ipv6 prefix-list PEER-A-PREFIXES seq 10 deny any
!
ipv6 prefix-list PEER-B-PREFIXES seq  5 permit 2001:0DB8:BBBB::/48 ge 64 le 64
ipv6 prefix-list PEER-B-PREFIXES seq 10 deny any
!
ipv6 prefix-list PEER-C-PREFIXES seq  5 permit 2001:0DB8:CCCC::/48 ge 64 le 64
ipv6 prefix-list PEER-C-PREFIXES seq 10 deny any
!
route-map RSCLIENT-A-IMPORT permit 10
  match peer 2001:0DB8::B
  call A-IMPORT-FROM-B
```

(continues on next page)

(continued from previous page)

```
route-map RSCLIENT-A-IMPORT permit 20
  match peer 2001:0DB8::C
  call A-IMPORT-FROM-C
!
route-map A-IMPORT-FROM-B permit 10
  match ipv6 address prefix-list COMMON-PREFIXES
  set metric 100
route-map A-IMPORT-FROM-B permit 20
  match ipv6 address prefix-list PEER-B-PREFIXES
  set community 65001:11111
!
route-map A-IMPORT-FROM-C permit 10
  match ipv6 address prefix-list COMMON-PREFIXES
  set metric 200
route-map A-IMPORT-FROM-C permit 20
  match ipv6 address prefix-list PEER-C-PREFIXES
  set community 65001:22222
!
route-map RSCLIENT-A-EXPORT permit 10
  match peer 2001:0DB8::B
  match ipv6 address prefix-list PEER-A-PREFIXES
route-map RSCLIENT-A-EXPORT permit 20
  match peer 2001:0DB8::C
  match ipv6 address prefix-list PEER-A-PREFIXES
!
...
...
...
```

If you compare the initial configuration of RA with the route server configuration above, you can see how easy it is to generate the Import and Export policies for RA from the In and Out route-maps of RA's original configuration.

When there was no route server, RA maintained two peerings, one with RB and another with RC. Each of this peerings had an In route-map configured. To build the Import route-map for client RA in the route server, simply add route-map entries following this scheme:

```
route-map <NAME> permit 10
    match peer <Peer Address>
    call <In Route-Map for this Peer>
route-map <NAME> permit 20
    match peer <Another Peer Address>
    call <In Route-Map for this Peer>
```

This is exactly the process that has been followed to generate the route-map RSCLIENT-A-IMPORT. The route-maps that are called inside it (A-IMPORT-FROM-B and A-IMPORT-FROM-C) are exactly the same than the In route-maps from the original configuration of RA (PEER-B-IN and PEER-C-IN), only the name is different.

The same could have been done to create the Export policy for RA (route-map RSCLIENT-A-EXPORT), but in this case the original Out route-maps where so simple that we decided not to use the *call WORD* commands, and we integrated all in a single route-map (RSCLIENT-A-EXPORT).

The Import and Export policies for RB and RC are not shown, but the process would be identical.

Further considerations about Import and Export route-maps

The current version of the route server patch only allows to specify a route-map for import and export policies, while in a standard BGP speaker apart from route-maps there are other tools for performing input and output filtering (access-lists, community-lists, …). But this does not represent any limitation, as all kinds of filters can be included in import/export route-maps. For example suppose that in the non-route-server scenario peer RA had the following filters configured for input from peer B:

```
neighbor 2001:0DB8::B prefix-list LIST-1 in
neighbor 2001:0DB8::B filter-list LIST-2 in
neighbor 2001:0DB8::B route-map PEER-B-IN in
...
...
route-map PEER-B-IN permit 10
  match ipv6 address prefix-list COMMON-PREFIXES
  set local-preference 100
route-map PEER-B-IN permit 20
  match ipv6 address prefix-list PEER-B-PREFIXES
  set community 65001:11111
```

It is possible to write a single route-map which is equivalent to the three filters (the community-list, the prefix-list and the route-map). That route-map can then be used inside the Import policy in the route server. Lets see how to do it:

```
neighbor 2001:0DB8::A route-map RSCLIENT-A-IMPORT in
...
!
...
route-map RSCLIENT-A-IMPORT permit 10
  match peer 2001:0DB8::B
  call A-IMPORT-FROM-B
...
...
!
route-map A-IMPORT-FROM-B permit 1
  match ipv6 address prefix-list LIST-1
  match as-path LIST-2
  on-match goto 10
route-map A-IMPORT-FROM-B deny 2
route-map A-IMPORT-FROM-B permit 10
  match ipv6 address prefix-list COMMON-PREFIXES
  set local-preference 100
route-map A-IMPORT-FROM-B permit 20
  match ipv6 address prefix-list PEER-B-PREFIXES
  set community 65001:11111
!
...
...
```

The route-map A-IMPORT-FROM-B is equivalent to the three filters (LIST-1, LIST-2 and PEER-B-IN). The first entry of route-map A-IMPORT-FROM-B (sequence number 1) matches if and only if both the prefix-list LIST-1 and the filter-list LIST-2 match. If that happens, due to the 'on-match goto 10' statement the next route-map entry to be processed will be number 10, and as of that point route-map A-IMPORT-FROM-B is identical to PEER-B-IN. If the first entry does not match, *on-match goto 10*' will be ignored and the next processed entry will be number 2, which will deny the route.

Thus, the result is the same that with the three original filters, i.e., if either LIST-1 or LIST-2 rejects the route, it does not reach the route-map PEER-B-IN. In case both LIST-1 and LIST-2 accept the route, it passes to PEER-B-IN, which can reject, accept or modify the route.

3.3.12 Prefix Origin Validation Using RPKI

Prefix Origin Validation allows BGP routers to verify if the origin AS of an IP prefix is legitimate to announce this IP prefix. The required attestation objects are stored in the Resource Public Key Infrastructure (RPKI). However, RPKI-enabled routers do not store cryptographic data itself but only validation information. The validation of the cryptographic data (so called Route Origin Authorization, or short ROA, objects) will be performed by trusted cache servers. The RPKI/RTR protocol defines a standard mechanism to maintain the exchange of the prefix/origin AS mapping between the cache server and routers. In combination with a BGP Prefix Origin Validation scheme a router is able to verify received BGP updates without suffering from cryptographic complexity.

The RPKI/RTR protocol is defined in **RFC 6810** and the validation scheme in **RFC 6811**. The current version of Prefix Origin Validation in FRR implements both RFCs.

For a more detailed but still easy-to-read background, we suggest:

- [Securing-BGP]
- [Resource-Certification]

Features of the Current Implementation

In a nutshell, the current implementation provides the following features

- The BGP router can connect to one or more RPKI cache servers to receive validated prefix to origin AS mappings. Advanced failover can be implemented by server sockets with different preference values.
- If no connection to an RPKI cache server can be established after a pre-defined timeout, the router will process routes without prefix origin validation. It still will try to establish a connection to an RPKI cache server in the background.
- By default, enabling RPKI does not change best path selection. In particular, invalid prefixes will still be considered during best path selection. However, the router can be configured to ignore all invalid prefixes.
- Route maps can be configured to match a specific RPKI validation state. This allows the creation of local policies, which handle BGP routes based on the outcome of the Prefix Origin Validation.
- Updates from the RPKI cache servers are directly applied and path selection is updated accordingly. (Soft reconfiguration **must** be enabled for this to work).

Enabling RPKI

You must install `frr-rpki-rtrlib` additional package for RPKI support, otherwise `bgpd` daemon won't startup.

`rpki`

This command enables the RPKI configuration mode. Most commands that start with *rpki* can only be used in this mode.

When it is used in a telnet session, leaving of this mode cause rpki to be initialized.

Executing this command alone does not activate prefix validation. You need to configure at least one reachable cache server. See section *Configuring RPKI/RTR Cache Servers* for configuring a cache server.

Remember to add `-M rpki` to the variable `bgpd_options` in `/etc/frr/daemons` , like so:

```
bgpd_options="   -A 127.0.0.1 -M rpki"
```

instead of the default setting:

```
bgpd_options="   -A 127.0.0.1"
```

Otherwise you will encounter an error when trying to enter RPKI configuration mode due to the `rpki` module not being loaded when the BGP daemon is initialized.

Examples of the error:

```
router(config)# debug rpki
% [BGP] Unknown command: debug rpki

router(config)# rpki
% [BGP] Unknown command: rpki
```

Note that the RPKI commands will be available in vtysh when running `find rpki` regardless of whether the module is loaded.

Configuring RPKI/RTR Cache Servers

The following commands are independent of a specific cache server.

`rpki polling_period (1-3600)`

Set the number of seconds the router waits until the router asks the cache again for updated data.

The default value is 300 seconds.

`rpki expire_interval (600-172800)`

Set the number of seconds the router waits until the router expires the cache.

The default value is 7200 seconds.

`rpki retry_interval (1-7200)`

Set the number of seconds the router waits until retrying to connect to the cache server.

The default value is 600 seconds.

`rpki cache (A.B.C.D|WORD) PORT [SSH_USERNAME] [SSH_PRIVKEY_PATH] [KNOWN_HOSTS_PATH] [source A.B.C.D] preference (1-255)`

Add a cache server to the socket. By default, the connection between router and cache server is based on plain TCP. Protecting the connection between router and cache server by SSH is optional. Deleting a socket removes the associated cache server and terminates the existing connection.

A.B.C.D|WORD Address of the cache server.

PORT Port number to connect to the cache server

SSH_USERNAME SSH username to establish an SSH connection to the cache server.

SSH_PRIVKEY_PATH Local path that includes the private key file of the router.

KNOWN_HOSTS_PATH Local path that includes the known hosts file. The default value depends on the configuration of the operating system environment, usually `~/.ssh/known_hosts`.

source A.B.C.D Source address of the RPKI connection to access cache server.

Validating BGP Updates

match rpki notfound|invalid|valid

Create a clause for a route map to match prefixes with the specified RPKI state.

In the following example, the router prefers valid routes over invalid prefixes because invalid routes have a lower local preference.

```
! Allow for invalid routes in route selection process
route bgp 60001
!
! Set local preference of invalid prefixes to 10
route-map rpki permit 10
 match rpki invalid
 set local-preference 10
!
! Set local preference of valid prefixes to 500
route-map rpki permit 500
 match rpki valid
 set local-preference 500
```

match rpki-extcommunity notfound|invalid|valid

Create a clause for a route map to match prefixes with the specified RPKI state, that is derived from the Origin Validation State extended community attribute (OVS). OVS extended community is non-transitive and is exchanged only between iBGP peers.

Debugging

debug rpki

Enable or disable debugging output for RPKI.

Displaying RPKI

show rpki prefix <A.B.C.D/M|X:X::X:X/M> [(1-4294967295)] [json]

Display validated prefixes received from the cache servers filtered by the specified prefix.

show rpki as-number ASN [json]

Display validated prefixes received from the cache servers filtered by ASN.

show rpki prefix-table [json]

Display all validated prefix to origin AS mappings/records which have been received from the cache servers and stored in the router. Based on this data, the router validates BGP Updates.

show rpki cache-server [json]

Display all configured cache servers, whether active or not.

show rpki cache-connection [json]

Display all cache connections, and show which is connected or not.

show bgp [afi] [safi] <A.B.C.D|A.B.C.D/M|X:X::X:X|X:X::X:X/M> rpki <valid|invalid|notfound>

Display for the specified prefix or address the bgp paths that match the given rpki state.

show bgp [afi] [safi] rpki <valid|invalid|notfound>

Display all prefixes that match the given rpki state.

RPKI Configuration Example

```
hostname bgpd1
password zebra
! log stdout
debug bgp updates
debug bgp keepalives
debug rpki
!
rpki
 rpki polling_period 1000
 rpki timeout 10
  ! SSH Example:
  rpki cache example.com source 141.22.28.223 22 rtr-ssh ./ssh_key/id_rsa ./ssh_key/id_
↪rsa.pub preference 1
  ! TCP Example:
  rpki cache rpki-validator.realmv6.org 8282 preference 2
  exit
!
router bgp 60001
 bgp router-id 141.22.28.223
 network 192.168.0.0/16
 neighbor 123.123.123.0 remote-as 60002
 neighbor 123.123.123.0 route-map rpki in
 neighbor 123.123.123.0 update-source 141.22.28.223
!
 address-family ipv6
  neighbor 123.123.123.0 activate
  neighbor 123.123.123.0 route-map rpki in
 exit-address-family
!
route-map rpki permit 10
 match rpki invalid
 set local-preference 10
!
route-map rpki permit 20
 match rpki notfound
 set local-preference 20
!
route-map rpki permit 30
 match rpki valid
 set local-preference 30
!
route-map rpki permit 40
!
```

3.3.13 Weighted ECMP using BGP link bandwidth

Overview

In normal equal cost multipath (ECMP), the route to a destination has multiple next hops and traffic is expected to be equally distributed across these next hops. In practice, flow-based hashing is used so that all traffic associated with a particular flow uses the same next hop, and by extension, the same path across the network.

Weighted ECMP using BGP link bandwidth introduces support for network-wide unequal cost multipathing (UCMP) to an IP destination. The unequal cost load balancing is implemented by the forwarding plane based on the weights associated with the next hops of the IP prefix. These weights are computed based on the bandwidths of the corresponding multipaths which are encoded in the `BGP link bandwidth extended community` as specified in [Draft-IETF-idr-link-bandwidth]. Exchange of an appropriate BGP link bandwidth value for a prefix across the network results in network-wide unequal cost multipathing.

One of the primary use cases of this capability is in the data center when a service (represented by its anycast IP) has an unequal set of resources across the regions (e.g., PODs) of the data center and the network itself provides the load balancing function instead of an external load balancer. Refer to [Draft-IETF-mohanty-bess-ebgp-dmz] and **RFC 7938** for details on this use case. This use case is applicable in a pure L3 network as well as in a EVPN network.

The traditional use case for BGP link bandwidth to load balance traffic to the exit routers in the AS based on the bandwidth of their external eBGP peering links is also supported.

Design Principles

Next hop weight computation and usage

As described, in UCMP, there is a weight associated with each next hop of an IP prefix, and traffic is expected to be distributed across the next hops in proportion to their weight. The weight of a next hop is a simple factoring of the bandwidth of the corresponding path against the total bandwidth of all multipaths, mapped to the range 1 to 100. What happens if not all the paths in the multipath set have link bandwidth associated with them? In such a case, in adherence to [Draft-IETF-idr-link-bandwidth], the behavior reverts to standard ECMP among all the multipaths, with the link bandwidth being effectively ignored.

Note that there is no change to either the BGP best path selection algorithm or to the multipath computation algorithm; the mapping of link bandwidth to weight happens at the time of installation of the route in the RIB.

If data forwarding is implemented by means of the Linux kernel, the next hop's weight is used in the hash calculation. The kernel uses the Hash threshold algorithm and use of the next hop weight is built into it; next hops need not be expanded to achieve UCMP. UCMP for IPv4 is available in older Linux kernels too, while UCMP for IPv6 is available from the 4.16 kernel onwards.

If data forwarding is realized in hardware, common implementations expand the next hops (i.e., they are repeated) in the ECMP container in proportion to their weight. For example, if the weights associated with 3 next hops for a particular route are 50, 25 and 25 and the ECMP container has a size of 16 next hops, the first next hop will be repeated 8 times and the other 2 next hops repeated 4 times each. Other implementations are also possible.

Unequal cost multipath across a network

For the use cases listed above, it is not sufficient to support UCMP on just one router (e.g., egress router), or individually, on multiple routers; UCMP must be deployed across the entire network. This is achieved by employing the BGP link-bandwidth extended community.

At the router which originates the BGP link bandwidth, there has to be user configuration to trigger it, which is described below. Receiving routers would use the received link bandwidth from their downstream routers to determine the next hop weight as described in the earlier section. Further, if the received link bandwidth is a transitive attribute, it would be propagated to eBGP peers, with the additional change that if the next hop is set to oneself, the cumulative link bandwidth of all downstream paths is propagated to other routers. In this manner, the entire network will know how to distribute traffic to an anycast service across the network.

The BGP link-bandwidth extended community is encoded in bytes-per-second. In the use case where UCMP must be based on the number of paths, a reference bandwidth of 1 Mbps is used. So, for example, if there are 4 equal cost paths to an anycast IP, the encoded bandwidth in the extended community will be 500,000. The actual value itself doesn't matter as long as all routers originating the link-bandwidth are doing it in the same way.

Configuration Guide

The configuration for weighted ECMP using BGP link bandwidth requires one essential step - using a route-map to inject the link bandwidth extended community. An additional option is provided to control the processing of received link bandwidth.

Injecting link bandwidth into the network

At the "entry point" router that is injecting the prefix to which weighted load balancing must be performed, a route-map must be configured to attach the link bandwidth extended community.

For the use case of providing weighted load balancing for an anycast service, this configuration will typically need to be applied at the TOR or Leaf router that is connected to servers which provide the anycast service and the bandwidth would be based on the number of multipaths for the destination.

For the use case of load balancing to the exit router, the exit router should be configured with the route map specifying the a bandwidth value that corresponds to the bandwidth of the link connecting to its eBGP peer in the adjoining AS. In addition, the link bandwidth extended community must be explicitly configured to be non-transitive.

The complete syntax of the route-map set command can be found at *BGP Extended Communities in Route Map*

This route-map is supported only at two attachment points: (a) the outbound route-map attached to a peer or peer-group, per address-family (b) the EVPN advertise route-map used to inject IPv4 or IPv6 unicast routes into EVPN as type-5 routes.

Since the link bandwidth origination is done by using a route-map, it can be constrained to certain prefixes (e.g., only for anycast services) or it can be generated for all prefixes. Further, when the route-map is used in the neighbor context, the link bandwidth usage can be constrained to certain peers only.

A sample configuration is shown below and illustrates link bandwidth advertisement towards the "SPINE" peer-group for anycast IPs in the range 192.168.x.x

```
ip prefix-list anycast_ip seq 10 permit 192.168.0.0/16 le 32
route-map anycast_ip permit 10
 match ip address prefix-list anycast_ip
 set extcommunity bandwidth num-multipaths
route-map anycast_ip permit 20
```

(continues on next page)

(continued from previous page)

```
!
router bgp 65001
 neighbor SPINE peer-group
 neighbor SPINE remote-as external
 neighbor 172.16.35.1 peer-group SPINE
 neighbor 172.16.36.1 peer-group SPINE
 !
 address-family ipv4 unicast
  network 110.0.0.1/32
  network 192.168.44.1/32
  neighbor SPINE route-map anycast_ip out
 exit-address-family
!
```

Controlling link bandwidth processing on the receiver

There is no configuration necessary to process received link bandwidth and translate it into the weight associated with the corresponding next hop; that happens by default. If some of the multipaths do not have the link bandwidth extended community, the default behavior is to revert to normal ECMP as recommended in [Draft-IETF-idr-link-bandwidth].

The operator can change these behaviors with the following configuration:

`bgp bestpath bandwidth <ignore | skip-missing | default-weight-for-missing>`

The different options imply behavior as follows:

- ignore: Ignore link bandwidth completely for route installation (i.e., do regular ECMP, not weighted)
- skip-missing: Skip paths without link bandwidth and do UCMP among the others (if at least some paths have link-bandwidth)
- default-weight-for-missing: Assign a low default weight (value 1) to paths not having link bandwidth

This configuration is per BGP instance similar to other BGP route-selection controls; it operates on both IPv4-unicast and IPv6-unicast routes in that instance. In an EVPN network, this configuration (if required) should be implemented in the tenant VRF and is again applicable for IPv4-unicast and IPv6-unicast, including the ones sourced from EVPN type-5 routes.

A sample snippet of FRR configuration on a receiver to skip paths without link bandwidth and do weighted ECMP among the other paths (if some of them have link bandwidth) is as shown below.

```
router bgp 65021
 bgp bestpath as-path multipath-relax
 bgp bestpath bandwidth skip-missing
 neighbor LEAF peer-group
 neighbor LEAF remote-as external
 neighbor 172.16.35.2 peer-group LEAF
 neighbor 172.16.36.2 peer-group LEAF
 !
 address-family ipv4 unicast
  network 130.0.0.1/32
 exit-address-family
!
```

Stopping the propagation of the link bandwidth outside a domain

The link bandwidth extended community will get automatically propagated with the prefix to EBGP peers, if it is encoded as a transitive attribute by the originator. If this propagation has to be stopped outside of a particular domain (e.g., stopped from being propagated to routers outside of the data center core network), the mechanism available is to disable the advertisement of all BGP extended communities on the specific peering/s. In other words, the propagation cannot be blocked just for the link bandwidth extended community. The configuration to disable all extended communities can be applied to a peer or peer-group (per address-family).

Of course, the other common way to stop the propagation of the link bandwidth outside the domain is to block the prefixes themselves from being advertised and possibly, announce only an aggregate route. This would be quite common in a EVPN network.

BGP link bandwidth and UCMP monitoring & troubleshooting

Existing operational commands to display the BGP routing table for a specific prefix will show the link bandwidth extended community also, if present.

An example of an IPv4-unicast route received with the link bandwidth attribute from two peers is shown below:

```
CLI# show bgp ipv4 unicast 192.168.10.1/32
BGP routing table entry for 192.168.10.1/32
Paths: (2 available, best #2, table default)
  Advertised to non peer-group peers:
  l1(swp1) l2(swp2) l3(swp3) l4(swp4)
  65002
    fe80::202:ff:fe00:1b from l2(swp2) (110.0.0.2)
    (fe80::202:ff:fe00:1b) (used)
      Origin IGP, metric 0, valid, external, multipath, bestpath-from-AS 65002
      Extended Community: LB:65002:125000000 (1000.000 Mbps)
      Last update: Thu Feb 20 18:34:16 2020

  65001
    fe80::202:ff:fe00:15 from l1(swp1) (110.0.0.1)
    (fe80::202:ff:fe00:15) (used)
      Origin IGP, metric 0, valid, external, multipath, bestpath-from-AS 65001, best␣
↪(Older Path)
      Extended Community: LB:65001:62500000 (500.000 Mbps)
      Last update: Thu Feb 20 18:22:34 2020
```

The weights associated with the next hops of a route can be seen by querying the RIB for a specific route.

For example, the next hop weights corresponding to the link bandwidths in the above example is illustrated below:

```
spine1# show ip route 192.168.10.1/32
Routing entry for 192.168.10.1/32
  Known via "bgp", distance 20, metric 0, best
  Last update 00:00:32 ago
  * fe80::202:ff:fe00:1b, via swp2, weight 66
  * fe80::202:ff:fe00:15, via swp1, weight 33
```

For troubleshooting, existing debug logs `debug bgp updates`, `debug bgp bestpath <prefix>`, `debug bgp zebra` and `debug zebra kernel` can be used.

A debug log snippet when `debug bgp zebra` is enabled and a route is installed by BGP in the RIB with next hop weights is shown below:

```
2020-02-29T06:26:19.927754+00:00 leaf1 bgpd[5459]: bgp_zebra_announce: p=192.168.150.1/
↪32, bgp_is_valid_label: 0
2020-02-29T06:26:19.928096+00:00 leaf1 bgpd[5459]: Tx route add VRF 33 192.168.150.1/32␣
↪metric 0 tag 0 count 2
2020-02-29T06:26:19.928289+00:00 leaf1 bgpd[5459]:   nhop [1]: 110.0.0.6 if 35 VRF 33 wt␣
↪50   RMAC 0a:11:2f:7d:35:20
2020-02-29T06:26:19.928479+00:00 leaf1 bgpd[5459]:   nhop [2]: 110.0.0.5 if 35 VRF 33 wt␣
↪50   RMAC 32:1e:32:a3:6c:bf
2020-02-29T06:26:19.928668+00:00 leaf1 bgpd[5459]: bgp_zebra_announce: 192.168.150.1/32:␣
↪announcing to zebra (recursion NOT set)
```

References

3.3.14 Flowspec

Overview

Flowspec introduces a new NLRI (Network Layer Reachability Information) encoding format that is used to distribute traffic rule flow specifications. Basically, instead of simply relying on destination IP address for IP prefixes, the IP prefix is replaced by a n-tuple consisting of a rule. That rule can be a more or less complex combination of the following:

- Network source/destination (can be one or the other, or both).
- Layer 4 information for UDP/TCP: source port, destination port, or any port.
- Layer 4 information for ICMP type and ICMP code.
- Layer 4 information for TCP Flags.
- Layer 3 information: DSCP value, Protocol type, packet length, fragmentation.
- Misc layer 4 TCP flags.

Note that if originally Flowspec defined IPv4 rules, this is also possible to use IPv6 address-family. The same set of combinations as defined for IPv4 can be used.

A combination of the above rules is applied for traffic filtering. This is encoded as part of specific BGP extended communities and the action can range from the obvious rerouting (to nexthop or to separate VRF) to shaping, or discard.

The following IETF drafts and RFCs have been used to implement FRR Flowspec:

- **RFC 5575**
- [Draft-IETF-IDR-Flowspec-redirect-IP]
- [Draft-IETF-IDR-Flow-Spec-V6]

Design Principles

FRR implements the Flowspec client side, that is to say that BGP is able to receive Flowspec entries, but is not able to act as manager and send Flowspec entries.

Linux provides the following mechanisms to implement policy based routing:

- Filtering the traffic with `Netfilter`. `Netfilter` provides a set of tools like `ipset` and `iptables` that are powerful enough to be able to filter such Flowspec filter rule.
- using non standard routing tables via `iproute2` (via the `ip rule` command provided by `iproute2`). `iproute2` is already used by FRR's *PBR* daemon which provides basic policy based routing based on IP source and destination criterion.

Below example is an illustration of what Flowspec will inject in the underlying system:

```
# linux shell
ipset create match0x102 hash:net,net counters
ipset add match0x102 32.0.0.0/16,40.0.0.0/16
iptables -N match0x102 -t mangle
iptables -A match0x102 -t mangle -j MARK --set-mark 102
iptables -A match0x102 -t mangle -j ACCEPT
iptables -i ntfp3 -t mangle -I PREROUTING -m set --match-set match0x102
             src,dst -g match0x102
ip rule add fwmark 102 lookup 102
ip route add 40.0.0.0/16 via 44.0.0.2 table 102
```

For handling an incoming Flowspec entry, the following workflow is applied:

- Incoming Flowspec entries are handled by *bgpd*, stored in the BGP RIB.
- Flowspec entry is installed according to its complexity.

It will be installed if one of the following filtering action is seen on the BGP extended community: either redirect IP, or redirect VRF, in conjunction with rate option, for redirecting traffic. Or rate option set to 0, for discarding traffic.

According to the degree of complexity of the Flowspec entry, it will be installed in *zebra* RIB. For more information about what is supported in the FRR implementation as rule, see *Limitations / Known Issues* chapter. Flowspec entry is split in several parts before being sent to *zebra*.

- *zebra* daemon receives the policy routing configuration

Policy Based Routing entities necessary to policy route the traffic in the underlying system, are received by *zebra*. Two filtering contexts will be created or appended in `Netfilter`: `ipset` and `iptable` context. The former is used to define an IP filter based on multiple criterium. For instance, an ipset `net:net` is based on two ip addresses, while `net,port,net` is based on two ip addresses and one port (for ICMP, UDP, or TCP). The way the filtering is used (for example, is src port or dst port used?) is defined by the latter filtering context. `iptable` command will reference the `ipset` context and will tell how to filter and what to do. In our case, a marker will be set to indicate `iproute2` where to forward the traffic to. Sometimes, for dropping action, there is no need to add a marker; the `iptable` will tell to drop all packets matching the `ipset` entry.

Configuration Guide

In order to configure an IPv4 Flowspec engine, use the following configuration. As of today, it is only possible to configure Flowspec on the default VRF.

```
router bgp <AS>
  neighbor <A.B.C.D> remote-as <remoteAS>
  neighbor <A:B::C:D> remote-as <remoteAS2>
  address-family ipv4 flowspec
   neighbor <A.B.C.D> activate
  exit
  address-family ipv6 flowspec
   neighbor <A:B::C:D> activate
  exit
exit
```

You can see Flowspec entries, by using one of the following show commands:

`show bgp ipv4 flowspec [detail | A.B.C.D]`

`show bgp ipv6 flowspec [detail | A:B::C:D]`

Per-interface configuration

One nice feature to use is the ability to apply Flowspec to a specific interface, instead of applying it to the whole machine. Despite the following IETF draft [Draft-IETF-IDR-Flowspec-Interface-Set] is not implemented, it is possible to manually limit Flowspec application to some incoming interfaces. Actually, not using it can result to some unexpected behaviour like accounting twice the traffic, or slow down the traffic (filtering costs). To limit Flowspec to one specific interface, use the following command, under *flowspec address-family* node.

`local-install <IFNAME | any>`

By default, Flowspec is activated on all interfaces. Installing it to a named interface will result in allowing only this interface. Conversely, enabling any interface will flush all previously configured interfaces.

VRF redirection

Another nice feature to configure is the ability to redirect traffic to a separate VRF. This feature does not go against the ability to configure Flowspec only on default VRF. Actually, when you receive incoming BGP flowspec entries on that default VRF, you can redirect traffic to an other VRF.

As a reminder, BGP flowspec entries have a BGP extended community that contains a Route Target. Finding out a local VRF based on Route Target consists in the following:

- A configuration of each VRF must be done, with its Route Target set Each VRF is being configured within a BGP VRF instance with its own Route Target list. Route Target accepted format matches the following: `A.B.C.D:U16`, or `U16:U32`, `U32:U16`.
- The first VRF with the matching Route Target will be selected to route traffic to. Use the following command under ipv4 unicast address-family node

`rt redirect import RTLIST...`

In order to illustrate, if the Route Target configured in the Flowspec entry is `E.F.G.H:II`, then a BGP VRF instance with the same Route Target will be set set. That VRF will then be selected. The below full configuration example depicts how Route Targets are configured and how VRFs and cross VRF configuration is done. Note that the VRF