FRR User Manual

Release latest

FRR

Sep 23, 2023

CONTENTS

CHAPTER

ONE

INTRODUCTION

1.1 Overview

FRR is a fully featured, high performance, free software IP routing suite.

FRR implements all standard routing protocols such as BGP, RIP, OSPF, IS-IS and more (see *Feature Matrix*), as well as many of their extensions.

FRR is a high performance suite written primarily in C. It can easily handle full Internet routing tables and is suitable for use on hardware ranging from cheap SBCs to commercial grade routers. It is actively used in production by hundreds of companies, universities, research labs and governments.

FRR is distributed under GPLv2, with development modeled after the Linux kernel. Anyone may contribute features, bug fixes, tools, documentation updates, or anything else.

FRR is a fork of Quagga.

1.1.1 How to get FRR

The official FRR website is located at https://frrouting.org/ and contains further information, as well as links to additional resources.

Several distributions provide packages for FRR. Check your distribution's repositories to find out if a suitable version is available.

Up-to-date Debian & Redhat packages are available at https://deb.frrouting.org/ & https://rpm.frrouting.org/ respectively.

For instructions on installing from source, refer to the developer documentation.

1.1.2 About FRR

FRR provides IP routing services. Its role in a networking stack is to exchange routing information with other routers, make routing and policy decisions, and inform other layers of these decisions. In the most common scenario, FRR installs routing decisions into the OS kernel, allowing the kernel networking stack to make the corresponding forwarding decisions.

In addition to dynamic routing FRR supports the full range of L3 configuration, including static routes, addresses, router advertisements etc. It has some light L2 functionality as well, but this is mostly left to the platform. This makes it suitable for deployments ranging from small home networks with static routes to Internet exchanges running full Internet tables.

FRR runs on all modern *NIX operating systems, including Linux and the BSDs. Feature support varies by platform; see the *Feature Matrix*.

System Requirements

System resources needed by FRR are highly dependent on workload. Routing software performance is particularly susceptible to external factors such as:

- Kernel networking stack
- Physical NIC
- Peer behavior
- Routing information scale

Because of these factors - especially the last one - it's difficult to lay out resource requirements.

To put this in perspective, FRR can be run on very low resource systems such as SBCs, provided it is not stressed too much. If you want to set up 4 Raspberry Pis to play with BGP or OSPF, it should work fine. If you ask a FRR to process a complete internet routing table on a Raspberry Pi, you will be disappointed. However, given enough resources, FRR ought to be capable of acting as a core IX router. Such a use case requires at least 4gb of memory and a recent quad-core server processor at a minimum.

If you are new to networking, an important thing to remember is that FRR is control plane software. It does not itself forward packets - it exchanges information with peers about how to forward packets. Forwarding plane performance largely depends on choice of NIC / ASIC.

System Architecture

Traditional routing software is made as a one process program which provides all of the routing protocol functionalities. FRR takes a different approach. FRR is a suite of daemons that work together to build the routing table. Each major protocol is implemented in its own daemon, and these daemons talk to a middleman daemon (*zebra*), which is responsible for coordinating routing decisions and talking to the dataplane.

This architecture allows for high resiliency, since an error, crash or exploit in one protocol daemon will generally not affect the others. It is also flexible and extensible since the modularity makes it easy to implement new protocols and tie them into the suite. Additionally, each daemon implements a plugin system allowing new functionality to be loaded at runtime.

An illustration of the large scale architecture is given below.

```
+----+  +----+  +-----+  +----+  +----+  +----+  +-----+
|bgpd|  |ripd|  |ospfd|  |ldpd|  |pbrd|  |pimd|  |.....|
+----+  +----+  +-----+  +----+  +----+  +----+  +-----+
     |       |        |       |       |       |        |
+----v-------v--------v-------v-------v-------v--------v
|                                                      |
|                         Zebra                        |
|                                                      |
+------------------------------------------------------+
       |                    |                   |
       |                    |                   |
+------v------+   +---------v--------+   +------v------+
|             |   |                  |   |             |
| *NIX Kernel |   | Remote dataplane |   | ........... |
|             |   |                  |   |             |
+-------------+   +------------------+   +-------------+
```

All of the FRR daemons can be managed through a single integrated user interface shell called *vtysh*. *vtysh* connects to each daemon through a UNIX domain socket and then works as a proxy for user input. In addition to a unified frontend, *vtysh* also provides the ability to configure all the daemons using a single configuration file through the integrated configuration mode. This avoids the overhead of maintaining a separate configuration file for each daemon.

FRR is currently implementing a new internal configuration system based on YANG data models. When this work is completed, FRR will be a fully programmable routing stack.

Supported Platforms

Currently FRR supports GNU/Linux and BSD. Porting FRR to other platforms is not too difficult as platform dependent code should be mostly limited to the *Zebra* daemon. Protocol daemons are largely platform independent. Please let us know if you can get FRR to run on a platform which is not listed below:

- GNU/Linux
- FreeBSD
- NetBSD
- OpenBSD

Versions of these platforms that are older than around 2 years from the point of their original release (in case of GNU/Linux, this is since the kernel's release on https://kernel.org/) may need some work. Similarly, the following platforms may work with some effort:

- MacOS

Recent versions of the following compilers are well tested:

- GNU's GCC
- LLVM's Clang
- Intel's ICC

Unsupported Platforms

In General if the platform you are attempting to use is not listed above then FRR does not support being run on that platform. The only caveat here is that version 7.5 and before Solaris was supported in a limited fashion.

Feature Matrix

The following table lists all protocols cross-referenced to all operating systems that have at least CI build tests. Note that for features, only features with system dependencies are included here; if you don't see the feature you're interested in, it should be supported on your platform.

Daemon / Feature	Linux	OpenBSD	FreeBSD	NetBSD
FRR Core				
zebra	Y	Y	Y	Y
VRF	4.8	N	N	N
MPLS	4.5	Y	N	N
pbrd (Policy Routing)	Y	N	N	N
WAN / Carrier protocols				
bgpd (BGP)	Y	Y	Y	Y
VRF / L3VPN	4.8 †4.3	CP	CP	CP
EVPN	4.18 †4.9	CP	CP	CP

continues on next page

Table 1 – continued from previous page

Daemon / Feature	Linux	OpenBSD	FreeBSD	NetBSD
VNC (Virtual Network Control)	CP	CP	CP	CP
Flowspec	CP	CP	CP	CP
ldpd (LDP)	4.5	Y	N	N
VPWS / PW	N	5.8	N	N
VPLS	N	5.8	N	N
nhrpd (NHRP)	Y	N	N	N
Link-State Routing				
ospfd (OSPFv2)	Y	Y	Y	Y
Segment Routing	4.12	N	N	N
ospf6d (OSPFv3)	Y	Y	Y	Y
isisd (IS-IS)	Y	Y	Y	Y
Distance-Vector Routing				
ripd (RIPv2)	Y	Y	Y	Y
ripngd (RIPng)	Y	Y	Y	Y
babeld (BABEL)	Y	Y	Y	Y
eigrpd (EIGRP)	Y	Y	Y	Y
Multicast Routing				
pimd (PIM)	4.19	N	Y	Y
SSM (Source Specific)	Y	N	Y	Y
ASM (Any Source)	Y	N	N	N
EVPN BUM Forwarding	5.0	N	N	N
vrrpd (VRRP)	5.1	N	N	N

The indicators have the following semantics:

- Y - daemon/feature fully functional
- X.X - fully functional with kernel version X.X or newer
- †X.X - restricted functionality or impaired performance with kernel version X.X or newer
- CP - control plane only (i.e. BGP route server / route reflector)
- N - daemon/feature not supported by operating system

Known Kernel Issues

- Linux < 4.11

  v6 Route Replacement - Linux kernels before 4.11 can cause issues with v6 route deletion when you have ECMP routes installed into the kernel. This especially becomes apparent if the route is being transformed from one ECMP path to another.

Supported RFCs

FRR implements the following RFCs:

Note: This list is incomplete.

BGP

- **RFC 1771** *A Border Gateway Protocol 4 (BGP-4). Y. Rekhter & T. Li. March 1995.*
- **RFC 1965** *Autonomous System Confederations for BGP. P. Traina. June 1996.*
- **RFC 1997** *BGP Communities Attribute. R. Chandra, P. Traina & T. Li. August 1996.*
- **RFC 1998** *An Application of the BGP Community Attribute in Multi-home Routing. E. Chen, T. Bates. August 1996.*
- **RFC 2385** *Protection of BGP Sessions via the TCP MD5 Signature Option. A. Heffernan. August 1998.*
- **RFC 2439** *BGP Route Flap Damping. C. Villamizar, R. Chandra, R. Govindan. November 1998.*
- **RFC 2545** *Use of BGP-4 Multiprotocol Extensions for IPv6 Inter-Domain Routing. P. Marques, F. Dupont. March 1999.*
- **RFC 2796** *BGP Route Reflection An alternative to full mesh IBGP. T. Bates & R. Chandrasekeran. June 1996.*
- **RFC 2842** *Capabilities Advertisement with BGP-4. R. Chandra, J. Scudder. May 2000.*
- **RFC 2858** *Multiprotocol Extensions for BGP-4. T. Bates, Y. Rekhter, R. Chandra, D. Katz. June 2000.*
- **RFC 2918** *Route Refresh Capability for BGP-4. E. Chen, September 2000.*
- **RFC 3107** *Carrying Label Information in BGP-4. Y. Rekhter & E. Rosen. May 2001.*
- **RFC 3765** *NOPEER Community for Border Gateway Protocol (BGP) Route Scope Control. G.Huston. April 2001.*
- **RFC 4271** *A Border Gateway Protocol 4 (BGP-4). Updates RFC1771. Y. Rekhter, T. Li & S. Hares. January 2006.*
- **RFC 4360** *BGP Extended Communities Attribute. S. Sangli, D. Tappan, Y. Rekhter. February 2006.*
- **RFC 4364** *BGP/MPLS IP Virtual Private Networks (VPNs). Y. Rekhter. February 2006.*
- **RFC 4456** *BGP Route Reflection An alternative to full mesh IBGP. T. Bates, E. Chen, R. Chandra. April 2006.*
- **RFC 4486** *Subcodes for BGP Cease Notification Message. E. Chen, V. Gillet. April 2006.*
- **RFC 4659** *BGP-MPLS IP Virtual Private Network (VPN) Extension for IPv6 VPN. J. De Clercq, D. Ooms, M. Carugi, F. Le Faucheur. September 2006.*
- **RFC 4724** *Graceful Restart Mechanism for BGP. S. Sangli, E. Chen, R. Fernando, J. Scudder, Y. Rekhter. January 2007.*
- **RFC 4760** *Multiprotocol Extensions for BGP-4. T. Bates, R. Chandra, D. Katz, Y. Rekhter. January 2007.*
- **RFC 4893** *BGP Support for Four-octet AS Number Space. Q. Vohra, E. Chen May 2007.*
- **RFC 5004** *Avoid BGP Best Path Transitions from One External to Another. E. Chen & S. Sangli. September 2007 (Partial support).*
- **RFC 5065** *Autonomous System Confederations for BGP. P. Traina, D. McPherson, J. Scudder. August 2007.*

- **RFC 5082** *The Generalized TTL Security Mechanism (GTSM). V. Gill, J. Heasley, D. Meyer, P. Savola, C. Pingnataro. October 2007.*
- **RFC 5291** *Outbound Route Filtering Capability. E. Chen, Y. Rekhter. August 2008.*
- **RFC 5292** *Address-Prefix-Based Outbound Route Filter for BGP-4. E. Chen, S. Sangli. August 2008.*
- **RFC 5492** *Capabilities Advertisement with BGP-4. J. Scudder, R. Chandra. February 2009.*
- **RFC 5575** *Dissemination of Flow Specification Rules. P. Marques, N. Sheth, R. Raszuk, B. Greene, J. Mauch, D. McPherson. August 2009.*
- **RFC 5668** *4-Octet AS Specific BGP Extended Community. Y. Rekhter, S. Sangli, D. Tappan October 2009.*
- **RFC 6286** *Autonomous-System-Wide Unique BGP Identifier for BGP-4. E. Chen, J. Yuan. June 2011.*
- **RFC 6472** *Recommendation for Not Using AS_SET and AS_CONFED_SET in BGP. W. Kumari, K. Sriram. December 2011.*
- **RFC 6608** *Subcodes for BGP Finite State Machine Error. J. Dong, M. Chen, Huawei Technologies, A. Suryanarayana, Cisco Systems. May 2012.*
- **RFC 6810** *The Resource Public Key Infrastructure (RPKI) to Router Protocol. R. Bush, R. Austein. January 2013.*
- **RFC 6811** *BGP Prefix Origin Validation. P. Mohapatra, J. Scudder, D. Ward, R. Bush, R. Austein. January 2013.*
- **RFC 6938** *Deprecation of BGP Path Attributes: DPA, ADVERTISER, and RCID_PATH / CLUSTER_ID. J. Scudder. May 2013.*
- **RFC 6996** *Autonomous System (AS) Reservation for Private Use. J. Mitchell. July 2013.*
- **RFC 7196** *Making Route Flap Damping Usable. C. Pelsser, R. Bush, K. Patel, P. Mohapatra, O. Maennel. May 2014.*
- **RFC 7300** *Reservation of Last Autonomous System (AS) Numbers. J. Haas, J. Mitchell. July 2014.*
- **RFC 7313** *Enhanced Route Refresh Capability for BGP-4. K. Patel, E. Chen, B. Venkatachalapathy. July 2014.*
- **RFC 7606** *Revised Error Handling for BGP UPDATE Messages. E. Chen, J. Scudder, P. Mohapatra, K. Patel. August 2015.*
- **RFC 7607** *Codification of AS 0 Processing. W. Kumari, R. Bush, H. Schiller, K. Patel. August 2015.*
- **RFC 7611** *BGP ACCEPT_OWN Community Attribute. J. Uttaro, P. Mohapatra, D. Smith, R. Raszuk, J. Scudder. August 2015.*
- **RFC 7911** *Advertisement of Multiple Paths in BGP. D. Walton, A. Retana, E. Chen, J. Scudder. July 2016.*
- **RFC 7947** *Internet Exchange BGP Route Server. E. Jasinska, N. Hilliard, R. Raszuk, N. Bakker. September 2016.*
- **RFC 7999** *BLACKHOLE Community. T. King, C. Dietzel, J. Snijders, G. Doering, G. Hankins. October 2016.*
- **RFC 8050** *Multi-Threaded Routing Toolkit (MRT) Routing Information Export Format with BGP Additional Path Extensions. C. Petrie, T. King. May 2017.*
- **RFC 8092** *BGP Large Communities Attribute. J. Heitz, Ed., J. Snijders, Ed, K. Patel, I. Bagdonas, N. Hilliard. February 2017.*
- **RFC 8093** *Deprecation of BGP Path Attribute Values 30, 31, 129, 241, 242, and 243. J. Snijders. February 2017.*
- **RFC 8097** *BGP Prefix Origin Validation State Extended Community. P. Mohapatra, K. Patel, J. Scudder, D. Ward, R. Bush. March 2017.*

- **RFC 8195** *Use of BGP Large Communities. J. Snijders, J. Heasley, M. Schmidt. June 2017.*
- **RFC 8203** *BGP Administrative Shutdown Communication. J. Snijders, J. Heitz, J. Scudder. July 2017.*
- **RFC 8212** *Default External BGP (EBGP) Route Propagation Behavior without Policies. J. Mauch, J. Snijders, G. Hankins. July 2017.*
- **RFC 8277** *Using BGP to Bind MPLS Labels to Address Prefixes. E. Rosen. October 2017.*
- **RFC 8538** *Notification Message Support for BGP Graceful Restart. K. Patel, R. Fernando, J. Scudder, J. Haas. March 2019.*
- **RFC 8654** *Extended Message Support for BGP. R. Bush, K. Patel, D. Ward. October 2019.*
- **RFC 9003** *Extended BGP Administrative Shutdown Communication. J. Snijders, J. Heitz, J. Scudder, A. Azimov. January 2021.*
- **RFC 9012** *The BGP Tunnel Encapsulation Attribute. K. Patel, G. Van de Velde, S. Sangli, J. Scudder. April 2021.*
- **RFC 9072** *Extended Optional Parameters Length for BGP OPEN Message. E. Chen, J. Scudder. July 2021.*
- **RFC 9234** *Route Leak Prevention and Detection Using Roles in UPDATE and OPEN Messages. A. Azimov, E. Bogomazov, R. Bush, K. Patel, K. Sriram. May 2022.*
- **RFC 9384** *A BGP Cease NOTIFICATION Subcode for Bidirectional Forwarding Detection (BFD). J. Haas. March 2023.*

OSPF

- **RFC 2328** *OSPF Version 2. J. Moy. April 1998.*
- **RFC 2370** *The OSPF Opaque LSA Option R. Coltun. July 1998.*
- **RFC 3101** *The OSPF Not-So-Stubby Area (NSSA) Option P. Murphy. January 2003.*
- **RFC 2740** *OSPF for IPv6. R. Coltun, D. Ferguson, J. Moy. December 1999.*
- **RFC 3137** *OSPF Stub Router Advertisement, A. Retana, L. Nguyen, R. White, A. Zinin, D. McPherson. June 2001*

ISIS

RIP

- **RFC 1058** *Routing Information Protocol. C.L. Hedrick. Jun-01-1988.*
- **RFC 2082** *RIP-2 MD5 Authentication. F. Baker, R. Atkinson. January 1997.*
- **RFC 2453** *RIP Version 2. G. Malkin. November 1998.*
- **RFC 2080** *RIPng for IPv6. G. Malkin, R. Minnear. January 1997.*

PIM

BFD

- **RFC 5880** *Bidirectional Forwarding Detection (BFD), D. Katz, D. Ward. June 2010*
- **RFC 5881** *Bidirectional Forwarding Detection (BFD) for IPv4 and IPv6 (Single Hop), D. Katz, D. Ward. June 2010*
- **RFC 5882** *Generic Application of Bidirectional Forwarding Detection (BFD), D. Katz, D. Ward. June 2010*
- **RFC 5883** *Bidirectional Forwarding Detection (BFD) for Multihop Paths, D. Katz, D. Ward. June 2010*

MPLS

- **RFC 2858** *Multiprotocol Extensions for BGP-4. T. Bates, Y. Rekhter, R. Chandra, D. Katz. June 2000.*
- **RFC 4364** *BGP/MPLS IP Virtual Private Networks (VPNs). Y. Rekhter. Feb 2006.*
- **RFC 4447** *Pseudowire Setup and Maintenance Using the Label Distribution Protocol (LDP), L. Martini, E. Rosen, N. El-Aawar, T. Smith, and G. Heron. April 2006.*
- **RFC 4659** *BGP-MPLS IP Virtual Private Network (VPN) Extension for IPv6 VPN. J. De Clercq, D. Ooms, M. Carugi, F. Le Faucheur. September 2006*
- **RFC 4762** *Virtual Private LAN Service (VPLS) Using Label Distribution Protocol (LDP) Signaling, M. Lasserre and V. Kompella. January 2007.*
- **RFC 5036** *LDP Specification, L. Andersson, I. Minei, and B. Thomas. October 2007.*
- **RFC 5561** *LDP Capabilities, B. Thomas, K. Raza, S. Aggarwal, R. Aggarwal, and JL. Le Roux. July 2009.*
- **RFC 5918** *Label Distribution Protocol (LDP) 'Typed Wildcard' Forward Equivalence Class (FEC), R. Asati, I. Minei, and B. Thomas. August 2010.*
- **RFC 5919** *Signaling LDP Label Advertisement Completion, R. Asati, P. Mohapatra, E. Chen, and B. Thomas. August 2010.*
- **RFC 6667** *LDP 'Typed Wildcard' Forwarding Equivalence Class (FEC) for PWid and Generalized PWid FEC Elements, K. Raza, S. Boutros, and C. Pignataro. July 2012.*
- **RFC 6720** *The Generalized TTL Security Mechanism (GTSM) for the Label Distribution Protocol (LDP), C. Pignataro and R. Asati. August 2012.*
- **RFC 7552** *Updates to LDP for IPv6, R. Asati, C. Pignataro, K. Raza, V. Manral, and R. Papneja. June 2015.*

VRRP

- **RFC 3768** *Virtual Router Redundancy Protocol (VRRP). R. Hinden. April 2004.*
- **RFC 5798** *Virtual Router Redundancy Protocol (VRRP) Version 3 for IPv4 and IPv6. S. Nadas. June 2000.*

SNMP

When SNMP support is enabled, the following RFCs are also supported:

- **RFC 1227** *SNMP MUX protocol and MIB. M.T. Rose. May-01-1991.*
- **RFC 1657** *Definitions of Managed Objects for the Fourth Version of the Border Gateway Protocol (BGP-4) using SMIv2. S. Willis, J. Burruss, J. Chu, Editor. July 1994.*
- **RFC 1724** *RIP Version 2 MIB Extension. G. Malkin & F. Baker. November 1994.*
- **RFC 1850** *OSPF Version 2 Management Information Base. F. Baker, R. Coltun. November 1995.*
- **RFC 2741** *Agent Extensibility (AgentX) Protocol. M. Daniele, B. Wijnen. January 2000.*

1.1.3 Mailing Lists

Italicized lists are private.

Topic	List
Development	dev@lists.frrouting.org
Users & Operators	frog@lists.frrouting.org
Announcements	announce@lists.frrouting.org
Security	security@lists.frrouting.org
Technical Steering Committee	tsc@lists.frrouting.org

The Development list is used to discuss and document general issues related to project development and governance. The public Slack instance and weekly technical meetings provide a higher bandwidth channel for discussions. The results of such discussions are reflected in updates, as appropriate, to code (i.e., merges), GitHub issues tracked issues, and for governance or process changes, updates to the Development list and either this file or information posted at FRR.

1.1.4 Bug Reports

For information on reporting bugs, please see *Reporting Bugs*.

1.2 Installation

This section covers the basics of building, installing and setting up FRR.

1.2.1 From Packages

The project publishes packages for Red Hat, Centos, Debian and Ubuntu on the GitHub releases. page. External contributors offer packages for many other platforms including *BSD, Alpine, Gentoo, Docker, and others. There is currently no documentation on how to use those but we hope to add it soon.

1.2.2 From Snapcraft

In addition to traditional packages the project also builds and publishes universal Snap images, available at https://snapcraft.io/frr.

1.2.3 From Source

Building FRR from source is the best way to ensure you have the latest features and bug fixes. Details for each supported platform, including dependency package listings, permissions, and other gotchas, are in the developer's documentation. This section provides a brief overview on the process.

Getting the Source

FRR's source is available on the project GitHub page.

```
git clone https://github.com/FRRouting/frr.git
```

When building from Git there are several branches to choose from. The `master` branch is the primary development branch. It should be considered unstable. Each release has its own branch named `stable/X.X`, where `X.X` is the release version.

In addition, release tarballs are published on the GitHub releases page here.

Build Configuration

FRR has an excellent configure script which automatically detects most host configurations. There are several additional configure options to customize the build to include or exclude specific features and dependencies.

First, update the build system. Change into your FRR source directory and issue:

```
./bootstrap.sh
```

This will install any missing build scripts and update the Autotools configuration. Once this is done you can move on to choosing your configuration options from the list below.

`--enable-tcmalloc`
Enable the alternate malloc library. In some cases this is faster and more efficient, in some cases it is not.

`--disable-doc`
Do not build any documentation, including this one.

`--enable-doc-html`
From the documentation build html docs as well in addition to the normal output.

`--disable-zebra`
Do not build zebra daemon. This generally only be useful in a scenario where you are building bgp as a standalone server.

`--disable-ripd`
Do not build ripd.

`--disable-ripngd`
Do not build ripngd.

`--disable-ospfd`
Do not build ospfd.

`--disable-ospf6d`
: Do not build ospf6d.

`--disable-bgpd`
: Do not build bgpd.

`--disable-ldpd`
: Do not build ldpd.

`--disable-nhrpd`
: Do not build nhrpd.

`--disable-eigrpd`
: Do not build eigrpd.

`--disable-babeld`
: Do not build babeld.

`--disable-watchfrr`
: Do not build watchfrr. Watchfrr is used to integrate daemons into startup/shutdown software available on your machine. This is needed for systemd integration, if you disable watchfrr you cannot have any systemd integration.

`--enable-werror`
: Build with all warnings converted to errors as a compile option. This is recommended for developers only.

`--disable-pimd`
: Turn off building of pimd. On some BSD platforms pimd will not build properly due to lack of kernel support.

`--disable-vrrpd`
: Turn off building of vrrpd. Linux is required for vrrpd support; other platforms are not supported.

`--disable-pbrd`
: Turn off building of pbrd. This daemon currently requires linux in order to function properly.

`--enable-sharpd`
: Turn on building of sharpd. This daemon facilitates testing of FRR and can also be used as a quick and easy route generator.

`--disable-staticd`
: Do not build staticd. This daemon is necessary if you want static routes.

`--disable-bfdd`
: Do not build bfdd.

`--disable-bgp-announce`
: Make *bgpd* which does not make bgp announcements at all. This feature is good for using *bgpd* as a BGP announcement listener.

`--disable-bgp-vnc`
: Turn off bgpd's ability to use VNC.

`--disable-bgp-bmp`
: Turn off BGP BMP support

`--enable-datacenter`
: This option is deprecated as it is superseded by the *-F* (profile) command line option which allows adjusting the setting at startup rather than compile time.

  Enable system defaults to work as if in a Data Center. See defaults.h for what is changed by this configure option.

`--enable-snmp`
: Enable SNMP support. By default, SNMP support is disabled.

`--disable-ospfapi`
: Disable support for OSPF-API, an API to interface directly with ospfd. OSPF-API is enabled if –enable-opaque-lsa is set.

`--disable-ospfclient`
: Disable installation of the python ospfclient and building of the example OSPF-API client.

`--disable-isisd`
: Do not build isisd.

`--disable-fabricd`
: Do not build fabricd.

`--enable-isis-topology`
: Enable IS-IS topology generator.

`--enable-realms`
: Enable the support of Linux Realms. Convert tag values from 1-255 into a realm value when inserting into the Linux kernel. Then routing policy can be assigned to the realm. See the tc man page. This option is currently not compatible with the usage of nexthop groups in the linux kernel itself.

`--disable-irdp`
: Disable IRDP server support. This is enabled by default if we have both *struct in_pktinfo* and *struct icmphdr* available to us.

`--disable-rtadv`
: Disable support IPV6 router advertisement in zebra.

`--enable-gcc-rdynamic`
: Pass the `-rdynamic` option to the linker driver. This is in most cases necessary for getting usable backtraces. This option defaults to on if the compiler is detected as gcc, but giving an explicit enable/disable is suggested.

`--disable-backtrace`
: Controls backtrace support for the crash handlers. This is autodetected by default. Using the switch will enforce the requested behaviour, failing with an error if support is requested but not available. On BSD systems, this needs libexecinfo, while on glibc support for this is part of libc itself.

`--enable-dev-build`
: Turn on some options for compiling FRR within a development environment in mind. Specifically turn on -g3 -O0 for compiling options and add inclusion of grammar sandbox.

`--disable-snmp`
: Build without SNMP support.

`--disable-vtysh`
: Build without VTYSH.

`--enable-fpm`
: Build with FPM module support.

`--with-service-timeout=X`
: Set timeout value for FRR service. The time of restarting or reloading FRR service should not exceed this value. This number can be from 0-999. Additionally if this parameter is not passed or setting X = 0, FRR will take default value: 2 minutes.

`--enable-numeric-version`
: Alpine Linux does not allow non-numeric characters in the version string. With this option, we provide a way to strip out these characters for APK dev package builds.

`--disable-version-build-config`
: Remove the “configuerd with” field that has all of the build configuration arguments when reporting the version string in *show version* command.

`--with-pkg-extra-version=VER`
: Add extra version field, for packagers/distributions

`--with-pkg-git-version`
: Add git information to MOTD and build version string

`--enable-multipath=X`
: Compile FRR with up to X way ECMP supported. This number can be from 0-999. For backwards compatibility with older configure options when setting X = 0, we will build FRR with 64 way ECMP. This is needed because there are hardcoded arrays that FRR builds towards, so we need to know how big to make these arrays at build time. Additionally if this parameter is not passed in FRR will default to 16 ECMP.

`--enable-shell-access`
: Turn on the ability of FRR to access some shell options(telnet/ssh/bash/etc.) from vtysh itself. This option is considered extremely unsecure and should only be considered for usage if you really really know what you are doing. This option is deprecated and will be removed on Feb 1, 2024.

`--enable-gcov`
: Code coverage reports from gcov require adjustments to the C and LD flags. With this option, gcov instrumentation is added to the build and coverage reports are created during execution. The check-coverage make target is also created to ease report uploading to codecov.io. The upload requires the COMMIT (git hash) and TOKEN (codecov upload token) environment variables be set.

`--enable-config-rollbacks`
: Build with configuration rollback support. Requires SQLite3.

`--enable-confd=<dir>`
: Build the ConfD northbound plugin. Look for the libconfd libs and headers in *dir*.

`--enable-sysrepo`
: Build the Sysrepo northbound plugin.

`--enable-grpc`
: Enable the gRPC northbound plugin.

`--enable-zeromq`
: Enable the ZeroMQ handler.

`--with-libpam`
: Use libpam for PAM support in vtysh.

`--enable-time-check XXX`
: This option is deprecated as it was replaced by the `service cputime-stats` CLI command, which may be adjusted at runtime rather than being a compile-time setting. See there for further detail.

`--disable-cpu-time`
: This option is deprecated as it was replaced by the `service cputime-warning NNN` CLI command, which may be adjusted at runtime rather than being a compile-time setting. See there for further detail.

`--enable-pcreposix`
: Turn on the usage of PCRE Posix libs for regex functionality.

`--enable-pcre2posix`
: Turn on the usage of PCRE2 Posix libs for regex functionality.

  PCRE2 versions <= 10.31 work a bit differently. We suggest using at least >= 10.36.

`--enable-rpath`
: Set hardcoded rpaths in the executable [default=yes].

`--enable-scripting`
: Enable Lua scripting [default=no].

You may specify any combination of the above options to the configure script. By default, the executables are placed in `/usr/local/sbin` and the configuration files in `/usr/local/etc`. The `/usr/local/` installation prefix and other directories may be changed using the following options to the configuration script.

`--enable-ccls`
Enable the creation of a `.ccls` file in the top level source directory.

Some development environments (e.g., LSP server within emacs, et al.) can utilize `ccls` to provide highly sophisticated IDE features (e.g., semantically accurate jump-to definition/reference, and even code refactoring). The *–enable-ccls* causes `configure` to generate a configuration for the `ccls` command, based on the configured FRR build environment.

`--prefix` `<prefix>`
Install architecture-independent files in *prefix* [/usr/local].

`--sysconfdir` `<dir>`
Look for configuration files in *dir* [*prefix*/etc]. Note that sample configuration files will be installed here.

`--localstatedir` `<dir>`
Configure zebra to use *dir* for local state files, such as pid files and unix sockets.

`--with-scriptdir` `<dir>`
Look for Lua scripts in `dir` [`prefix`/etc/frr/scripts].

`--with-yangmodelsdir` `<dir>`
Look for YANG modules in *dir* [*prefix*/share/yang]. Note that the FRR YANG modules will be installed here.

`--with-vici-socket` `<path>`
Set StrongSWAN vici interface socket path [/var/run/charon.vici].

Note: The former `--enable-systemd` option does not exist anymore. Support for systemd is now always available through built-in functions, without depending on libsystemd.

Python dependency, documentation and tests

FRR's documentation and basic unit tests heavily use code written in Python. Additionally, FRR ships Python extensions written in C which are used during its build process.

To this extent, FRR needs the following:

- an installation of CPython, preferably version 3.2 or newer (2.7 works but is end of life and will stop working at some point.)
- development files (mostly headers) for that version of CPython
- an installation of *sphinx* for that version of CPython, to build the documentation
- an installation of *pytest* for that version of CPython, to run the unit tests

The *sphinx* and *pytest* dependencies can be avoided by not building documentation / not running `make check`, but the CPython dependency is a hard dependency of the FRR build process (for the *clippy* tool.)

Least-Privilege Support

Additionally, you may configure zebra to drop its elevated privileges shortly after startup and switch to another user. The configure script will automatically try to configure this support. There are three configure options to control the behaviour of FRR daemons.

`--enable-user` `<user>`
: Switch to user *user shortly after startup, and run as user `user* in normal operation.

`--enable-group` `<user>`
: Switch real and effective group to *group* shortly after startup.

`--enable-vty-group` `<group>`
: Create Unix Vty sockets (for use with vtysh) with group ownership set to *group*. This allows one to create a separate group which is restricted to accessing only the vty sockets, hence allowing one to delegate this group to individual users, or to run vtysh setgid to this group.

The default user and group which will be configured is 'frr' if no user or group is specified. Note that this user or group requires write access to the local state directory (see *`--localstatedir`*) and requires at least read access, and write access if you wish to allow daemons to write out their configuration, to the configuration directory (see *`--sysconfdir`*).

On systems which have the 'libcap' capabilities manipulation library (currently only Linux), FRR will retain only minimal capabilities required and will only raise these capabilities for brief periods. On systems without libcap, FRR will run as the user specified and only raise its UID to 0 for brief periods.

Linux Notes

There are several options available only to GNU/Linux systems. If you use GNU/Linux, make sure that the current kernel configuration is what you want. FRR will run with any kernel configuration but some recommendations do exist.

`CONFIG_NETLINK` Kernel/User Netlink socket. This enables an advanced interface between the Linux kernel and *zebra* (*Kernel Interface*).

`CONFIG_RTNETLINK` This makes it possible to receive Netlink routing messages. If you specify this option, *zebra* can detect routing information updates directly from the kernel (*Kernel Interface*).

`CONFIG_IP_MULTICAST` This option enables IP multicast and should be specified when you use *ripd* (*RIP*) or *ospfd* (*OSPFv2*) because these protocols use multicast.

Linux sysctl settings and kernel modules

There are several kernel parameters that impact overall operation of FRR when using Linux as a router. Generally these parameters should be set in a sysctl related configuration file, e.g., `/etc/sysctl.conf` on Ubuntu based systems and a new file `/etc/sysctl.d/90-routing-sysctl.conf` on Centos based systems. Additional kernel modules are also needed to support MPLS forwarding.

`IPv4 and IPv6 forwarding` The following are set to enable IP forwarding in the kernel:

```
net.ipv4.conf.all.forwarding=1
net.ipv6.conf.all.forwarding=1
```

`MPLS forwarding` Basic MPLS support was introduced in the kernel in version 4.1 and additional capability was introduced in 4.3 and 4.5. For some general information on Linux MPLS support, see https://www.netdevconf.org/1.1/proceedings/slides/prabhu-mpls-tutorial.pdf. The following modules should be loaded to support MPLS forwarding, and are generally added to a configuration file such as `/etc/modules-load.d/modules.conf`:

```
# Load MPLS Kernel Modules
mpls_router
mpls_iptunnel
```

The following is an example to enable MPLS forwarding in the kernel, typically by editing `/etc/sysctl.conf`:

```
# Enable MPLS Label processing on all interfaces
net.mpls.conf.eth0.input=1
net.mpls.conf.eth1.input=1
net.mpls.conf.eth2.input=1
net.mpls.platform_labels=100000
```

Make sure to add a line equal to `net.mpls.conf.<if>.input` for each interface '*<if>*' used with MPLS and to set labels to an appropriate value.

VRF forwarding General information on Linux VRF support can be found in https://www.kernel.org/doc/Documentation/networking/vrf.txt.

Kernel support for VRFs was introduced in 4.3, but there are known issues in versions up to 4.15 (for IPv4) and 5.0 (for IPv6). The FRR CI system doesn't perform VRF tests on older kernel versions, and VRFs may not work on them. If you experience issues with VRF support, you should upgrade your kernel version.

See also:

Virtual Routing and Forwarding

Building

Once you have chosen your configure options, run the configure script and pass the options you chose:

```
./configure \
    --prefix=/usr \
    --localstatedir=/var/run/frr \
    --sbindir=/usr/lib/frr \
    --sysconfdir=/etc/frr \
    --enable-pimd \
    --enable-watchfrr \
    ...
```

After configuring the software, you are ready to build and install it in your system.

```
make && sudo make install
```

If everything finishes successfully, FRR should be installed. You should now skip to the section on *Basic Setup*.

1.3 Basic Setup

After installing FRR, some basic configuration must be completed before it is ready to use.

1.3.1 Crash logs

If any daemon should crash for some reason (segmentation fault, assertion failure, etc.), it will attempt to write a backtrace to a file located in `/var/tmp/frr/<daemon>[-<instance>].<pid>/crashlog`. This feature is not affected by any configuration options.

The crashlog file's directory also contains files corresponding to per-thread message buffers in files named `/var/tmp/frr/<daemon>[-<instance>].<pid>/logbuf.<tid>`. In case of a crash, these may contain unwritten buffered log messages. To show the contents of these buffers, pipe their contents through `tr '\0' '\n'`. A blank line marks the end of valid unwritten data (it will generally be followed by garbled, older log messages since the buffer is not cleared.)

1.3.2 Daemons Configuration File

After a fresh install, starting FRR will do nothing. This is because daemons must be explicitly enabled by editing a file in your configuration directory. This file is usually located at `/etc/frr/daemons` and determines which daemons are activated when issuing a service start / stop command via init or systemd. The file initially looks like this:

```
zebra=no
bgpd=no
ospfd=no
ospf6d=no
ripd=no
ripngd=no
isisd=no
pimd=no
ldpd=no
nhrpd=no
eigrpd=no
babeld=no
sharpd=no
staticd=no
pbrd=no
bfdd=no
fabricd=no

#
# If this option is set the /etc/init.d/frr script automatically loads
# the config via "vtysh -b" when the servers are started.
# Check /etc/pam.d/frr if you intend to use "vtysh"!
#
vtysh_enable=yes
zebra_options=" -s 90000000 --daemon -A 127.0.0.1"
bgpd_options="   --daemon -A 127.0.0.1"
ospfd_options="  --daemon -A 127.0.0.1"
ospf6d_options=" --daemon -A ::1"
ripd_options="   --daemon -A 127.0.0.1"
ripngd_options=" --daemon -A ::1"
```

(continues on next page)

(continued from previous page)

```
isisd_options="  --daemon -A 127.0.0.1"
pimd_options="  --daemon -A 127.0.0.1"
ldpd_options="  --daemon -A 127.0.0.1"
nhrpd_options="  --daemon -A 127.0.0.1"
eigrpd_options="  --daemon -A 127.0.0.1"
babeld_options="  --daemon -A 127.0.0.1"
sharpd_options="  --daemon -A 127.0.0.1"
staticd_options="  --daemon -A 127.0.0.1"
pbrd_options="  --daemon -A 127.0.0.1"
bfdd_options="  --daemon -A 127.0.0.1"
fabricd_options="  --daemon -A 127.0.0.1"

#MAX_FDS=1024
# The list of daemons to watch is automatically generated by the init script.
#watchfrr_options=""

# for debugging purposes, you can specify a "wrap" command to start instead
# of starting the daemon directly, e.g. to use valgrind on ospfd:
#   ospfd_wrap="/usr/bin/valgrind"
# or you can use "all_wrap" for all daemons, e.g. to use perf record:
#   all_wrap="/usr/bin/perf record --call-graph -"
# the normal daemon command is added to this at the end.
```

Breaking this file down:

```
bgpd=yes
```

To enable a particular daemon, simply change the corresponding ‘no’ to ‘yes’. Subsequent service restarts should start the daemon.

```
vtysh_enable=yes
```

As the comment says, this causes *VTYSH* to apply configuration when starting the daemons. This is useful for a variety of reasons touched on in the VTYSH documentation and should generally be enabled.

```
MAX_FDS=1024
```

This allows the operator to control the number of open file descriptors each daemon is allowed to start with. The current assumed value on most operating systems is 1024. If the operator plans to run bgp with several thousands of peers then this is where we would modify FRR to allow this to happen.

```
FRR_NO_ROOT="yes"
```

This option allows you to run FRR as a non-root user. Use this option only when you know what you are doing since most of the daemons in FRR will not be able to run under a regular user. This option is useful for example when you run FRR in a container with a designated user instead of root.

```
zebra_options=" -s 90000000 --daemon -A 127.0.0.1"
bgpd_options="   --daemon -A 127.0.0.1"
...
```

The next set of lines controls what options are passed to daemons when started from the service script. Usually daemons will have `--daemon` and `-A <address>` specified in order to daemonize and listen for VTY commands on a particular address.

The remaining file content regarding *watchfrr_options* and **_wrap* settings should not normally be needed; refer to the comments in case they are.

1.3.3 Services

FRR daemons have their own terminal interface or VTY. After installation, it's a good idea to setup each daemon's port number to connect to them. To do this add the following entries to `/etc/services`.

```
zebrasrv      2600/tcp                  # zebra service
zebra         2601/tcp                  # zebra vty
ripd          2602/tcp                  # RIPd vty
ripngd        2603/tcp                  # RIPngd vty
ospfd         2604/tcp                  # OSPFd vty
bgpd          2605/tcp                  # BGPd vty
ospf6d        2606/tcp                  # OSPF6d vty
ospfapi       2607/tcp                  # ospfapi
isisd         2608/tcp                  # ISISd vty
babeld        2609/tcp                  # BABELd vty
nhrpd         2610/tcp                  # nhrpd vty
pimd          2611/tcp                  # PIMd vty
ldpd          2612/tcp                  # LDPd vty
eigprd        2613/tcp                  # EIGRPd vty
bfdd          2617/tcp                  # bfdd vty
fabricd       2618/tcp                  # fabricd vty
vrrpd         2619/tcp                  # vrrpd vty
```

If you use a FreeBSD newer than 2.2.8, the above entries are already added to `/etc/services` so there is no need to add it. If you specify a port number when starting the daemon, these entries may not be needed.

You may need to make changes to the config files in /etc/frr.

1.3.4 Systemd

Although not installed when installing from source, FRR provides a service file for use with `systemd`. It is located in `tools/frr.service` in the Git repository. If `systemctl status frr.service` indicates that the FRR service is not found, copy the service file from the Git repository into your preferred location. A good place is usually `/etc/systemd/system/`.

After issuing a `systemctl daemon-reload`, you should be able to start the FRR service via `systemctl start frr`. If this fails, or no daemons are started. check the `journalctl` logs for an indication of what went wrong.

1.3.5 Operations

This section covers a few common operational tasks and how to perform them.

Interactive Shell

FRR offers an IOS-like interactive shell called `vtysh` where a user can run individual configuration or show commands. To get into this shell, issue the `vtysh` command from either a privilege user (root, or with sudo) or a user account that is part of the `frrvty` group. e.g.

```
root@ub18:~# vtysh

Hello, this is FRRouting (version 8.1-dev).
Copyright 1996-2005 Kunihiro Ishiguro, et al.

ub18#
```

Note: The default install location for vtysh is /usr/bin/vtysh

Restarting

Restarting kills all running FRR daemons and starts them again. Any unsaved configuration will be lost.

```
service frr restart
```

Note: Alternatively, you can invoke the init script directly:

```
/etc/init.d/frr restart
```

Or, if using systemd:

```
systemctl restart frr
```

Reloading

Reloading applies the differential between on-disk configuration and the current effective configuration of running FRR processes. This includes starting daemons that were previously stopped and any changes made to individual or unified daemon configuration files.

```
service frr reload
```

Note: Alternatively, you can invoke the init script directly:

```
/etc/init.d/frr reload
```

Or, if using systemd:

```
systemctl reload frr
```

See *FRR-RELOAD* for more about the *frr-reload.py* script.

Starting a new daemon

Suppose *bgpd* and *zebra* are running, and you wish to start *pimd*. In `/etc/frr/daemons` make the following change:

```
- pimd=no
+ pimd=yes
```

Then perform a reload.

Currently there is no way to stop or restart an individual daemon. This is because FRR's monitoring program cannot currently distinguish between a crashed / killed daemon versus one that has been intentionally stopped or restarted. The closest that can be achieved is to remove all configuration for the daemon, and set its line in `/etc/frr/daemons` to `=no`. Once this is done, the daemon will be stopped the next time FRR is restarted.

Network Namespaces

It is possible to run FRR in different network namespaces so it can be further compartmentalized (e.g. confining to a smaller subset network). The network namespace configuration can be used in the default FRR configuration pathspace or it can be used in a different pathspace (*-N/–pathspace*).

To use FRR network namespace in the default pathspace you should add or uncomment the `watchfrr_options` line in `/etc/frr/daemons`:

```
- #watchfrr_options="--netns"
+ watchfrr_options="--netns=<network-namespace-name>"
```

If you want to use a different pathspace with the network namespace (the recommended way) you should add/uncomment the `watchfrr_options` line in `/etc/frr/<namespace>/daemons`:

```
- #watchfrr_options="--netns"
+ #watchfrr_options="--netns=<network-namespace-name>"
+
+ # `--netns` argument is optional and if not provided it will
+ # default to the pathspace name.
+ watchfrr_options="--netns"
```

To start FRR in the new pathspace+network namespace the initialization script should be called with an extra parameter:

```
/etc/init.d/frr start <pathspace-name>
```

Note: Some Linux distributions might not use the default init script shipped with FRR, in that case you might want to try running the bundled script in `/usr/lib/frr/frrinit.sh`.

On systemd you might create different units or parameterize the existing one. See the man page: https://www.freedesktop.org/software/systemd/man/systemd.unit.html

CHAPTER TWO

BASICS

2.1 Basic Commands

The following sections discuss commands common to all the routing daemons.

2.1.1 Config Commands

In a config file, you can write the debugging options, a vty's password, routing daemon configurations, a log file name, and so forth. This information forms the initial command set for a routing beast as it is starting.

Config files are generally found in /etc/frr.

Config Methods

There are two ways of configuring FRR.

Traditionally each of the daemons had its own config file. The daemon name plus `.conf` was the default config file name. For example, zebra's default config file was `zebra.conf`. This method is deprecated.

Because of the amount of config files this creates, and the tendency of one daemon to rely on others for certain functionality, most deployments now use "integrated" configuration. In this setup all configuration goes into a single file, typically `/etc/frr/frr.conf`. When starting up FRR using an init script or systemd, `vtysh` is invoked to read the config file and send the appropriate portions to only the daemons interested in them. Running configuration updates are persisted back to this single file using `vtysh`. This is the recommended method. To use this method, add the following line to `/etc/frr/vtysh.conf`:

```
service integrated-vtysh-config
```

If you installed from source or used a package, this is probably already present.

If desired, you can specify a config file using the *`-f`* or *`--config_file`* options when starting a daemon.

Basic Config Commands

`hostname HOSTNAME`
: Set hostname of the router. It is only for current `vtysh`, it will not be saved to any configuration file even with `write file`.

`domainname DOMAINNAME`
: Set domainname of the router. It is only for current `vtysh`, it will not be saved to any configuration file even with `write file`.

`password PASSWORD`
: Set password for vty interface. The `no` form of the command deletes the password. If there is no password, a vty won't accept connections.

`enable password PASSWORD`
: Set enable password. The `no` form of the command deletes the enable password.

`service cputime-stats`
: Collect CPU usage statistics for individual FRR event handlers and CLI commands. This is enabled by default and can be disabled if the extra overhead causes a noticeable slowdown on your system.

  Disabling these statistics will also make the *`service cputime-warning (1-4294967295)`* limit non-functional.

`service cputime-warning (1-4294967295)`
: Warn if the CPU usage of an event handler or CLI command exceeds the specified limit (in milliseconds.) Such warnings are generally indicative of some routine in FRR mistakenly blocking/hogging the processing loop and should be reported as a FRR bug.

  The default limit is 5 seconds (i.e. 5000), but this can be changed by the deprecated `--enable-time-check=...` compile-time option.

  This command has no effect if *`service cputime-stats`* is disabled.

`service walltime-warning (1-4294967295)`
: Warn if the total wallclock time spent handling an event or executing a CLI command exceeds the specified limit (in milliseconds.) This includes time spent waiting for I/O or other tasks executing and may produce excessive warnings if the system is overloaded. (This may still be useful to provide an immediate sign that FRR is not operating correctly due to externally caused starvation.)

  The default limit is 5 seconds as above, including the same deprecated `--enable-time-check=...` compile-time option.

`log trap LEVEL`
: These commands are deprecated and are present only for historical compatibility. The log trap command sets the current logging level for all enabled logging destinations, and it sets the default for all future logging commands that do not specify a level. The normal default logging level is debugging. The `no` form of the command resets the default level for future logging commands to debugging, but it does not change the logging level of existing logging destinations.

`log stdout LEVEL`
: Enable logging output to stdout. If the optional second argument specifying the logging level is not present, the default logging level (typically debugging) will be used. The `no` form of the command disables logging to stdout. The `LEVEL` argument must have one of these values: emergencies, alerts, critical, errors, warnings, notifications, informational, or debugging. Note that the existing code logs its most important messages with severity `errors`.

  Note: If `systemd` is in use and stdout is connected to systemd, FRR will automatically switch to `journald` extended logging for this target.

Warning: FRRouting uses the `writev()` system call to write log messages. This call is supposed to be atomic, but in reality this does not hold for pipes or terminals, only regular files. This means that in rare cases, concurrent log messages from distinct threads may get jumbled in terminal output. Use a log file and `tail -f` if this rare chance is inacceptable to your setup.

`log file [FILENAME [LEVEL]]`

If you want to log into a file, please specify `filename` as in this example:

```
log file /var/log/frr/bgpd.log informational
```

If the optional second argument specifying the logging level is not present, the default logging level (typically debugging, but can be changed using the deprecated `log trap` command) will be used. The `no` form of the command disables logging to a file.

`log syslog [LEVEL]`

Enable logging output to syslog. If the optional second argument specifying the logging level is not present, the default logging level (typically debugging, but can be changed using the deprecated `log trap` command) will be used. The `no` form of the command disables logging to syslog.

Note: This uses the system's `syslog()` API, which does not support message batching or structured key/value data pairs. If possible, use *`log extended EXTLOGNAME`* with *`destination syslog [supports-rfc5424]`* instead of this.

`log extended EXTLOGNAME`

Create an extended logging target with the specified name. The name has no further meaning and is only used to identify the target. Multiple targets can be created and deleted with the `no` form.

Refer to *Extended Logging Target* for further details and suboptions.

`log monitor [LEVEL]`

This command is deprecated and does nothing.

`log facility [FACILITY]`

This command changes the facility used in syslog messages. The default facility is `daemon`. The `no` form of the command resets the facility to the default `daemon` facility.

`log record-priority`

To include the severity in all messages logged to a file, to stdout, or to a terminal monitor (i.e. anything except syslog), use the `log record-priority` global configuration command. To disable this option, use the `no` form of the command. By default, the severity level is not included in logged messages. Note: some versions of syslogd can be configured to include the facility and level in the messages emitted.

`log timestamp precision [(0-6)]`

This command sets the precision of log message timestamps to the given number of digits after the decimal point. Currently, the value must be in the range 0 to 6 (i.e. the maximum precision is microseconds). To restore the default behavior (1-second accuracy), use the `no` form of the command, or set the precision explicitly to 0.

```
log timestamp precision 3
```

In this example, the precision is set to provide timestamps with millisecond accuracy.

`log commands`

This command enables the logging of all commands typed by a user to all enabled log destinations. The note that logging includes full command lines, including passwords. If the daemon startup option *–command-log-always* is used to start the daemon then this command is turned on by default and cannot be turned off and the [no] form of the command is dissallowed.

`log filtered-file [FILENAME [LEVEL]]`
Configure a destination file for filtered logs with the *`log filter-text WORD`* command.

`log filter-text WORD`
This command forces logs to be filtered on a specific string. A log message will only be printed if it matches on one of the filters in the log-filter table. The filter only applies to file logging targets configured with *`log filtered-file [FILENAME [LEVEL]]`*.

Note: Log filters help when you need to turn on debugs that cause significant load on the system (enabling certain debugs can bring FRR to a halt). Log filters prevent this but you should still expect a small performance hit due to filtering each of all those logs.

Note: This setting is not saved to `frr.conf` and not shown in `show running-config`. It is intended for ephemeral debugging purposes only.

`clear log filter-text`
This command clears all current filters in the log-filter table.

`log immediate-mode`
Use unbuffered output for log and debug messages; normally there is some internal buffering.

`log unique-id`
Include `[XXXXX-XXXXX]` log message unique identifier in the textual part of log messages. This is enabled by default, but can be disabled with `no log unique-id`. Please make sure the IDs are enabled when including logs for FRR bug reports.

The unique identifiers are automatically generated based on source code file name, format string (before filling out) and severity. They do not change "randomly", but some cleanup work may cause large chunks of ID changes between releases. The IDs always start with a letter, consist of letters and numbers (and a dash for readability), are case insensitive, and `I`, `L`, `O` & `U` are excluded.

This option will not affect future logging targets which allow putting the unique identifier in auxiliary metadata outside the log message text content. (No such logging target exists currently, but RFC5424 syslog and systemd's journald both support it.)

`debug unique-id XXXXX-XXXXX backtrace`
Print backtraces (call stack) for specific log messages, identified by their unique ID (see above.) Includes source code location and current event handler being executed. On some systems you may need to install a *debug symbols* package to get proper function names rather than raw code pointers.

This command can be issued inside and outside configuration mode, and is saved to configuration only if it was given in configuration mode.

Warning: Printing backtraces can significantly slow down logging calls and cause log files to quickly balloon in size. Remember to disable backtraces when they're no longer needed.

`debug routemap [detail]`
This command turns on debugging of routemaps. When detail is specified more data is provided to the operator about the reasoning about what is going on in the routemap code.

`service password-encryption`
Encrypt password.

`service advanced-vty`
Enable advanced mode VTY.

`service terminal-length (0-512)`
Set system wide line configuration. This configuration command applies to all VTY interfaces.

`line vty`
Enter vty configuration mode.

`banner motd default`
Set default motd string.

`banner motd file FILE`
Set motd string from file. The file must be in directory specified under `--sysconfdir`.

`banner motd line LINE`
Set motd string from an input.

`exec-timeout MINUTE [SECOND]`
Set VTY connection timeout value. When only one argument is specified it is used for timeout value in minutes. Optional second argument is used for timeout value in seconds. Default timeout value is 10 minutes. When timeout value is zero, it means no timeout.

Not setting this, or setting the values to 0 0, means a timeout will not be enabled.

`access-class ACCESS-LIST`
Restrict vty connections with an access list.

`allow-reserved-ranges`
Allow using IPv4 reserved (Class E) IP ranges for daemons. E.g.: setting IPv4 addresses for interfaces or allowing reserved ranges in BGP next-hops.

If you need multiple FRR instances (or FRR + any other daemon) running in a single router and peering via 127.0.0.0/8, it's also possible to use this knob if turned on.

Default: off.

Sample Config File

Below is a sample configuration file for the zebra daemon.

```
!
! Zebra configuration file
!
frr version 6.0
frr defaults traditional
!
hostname Router
password zebra
enable password zebra
!
log stdout
!
!
```

`!` and `#` are comment characters. If the first character of the word is one of the comment characters then from the rest of the line forward will be ignored as a comment.

```
password zebra!password
```

If a comment character is not the first character of the word, it's a normal character. So in the above example `!` will not be regarded as a comment and the password is set to `zebra!password`.

Configuration versioning, profiles and upgrade behavior

All frr daemons share a mechanism to specify a configuration profile and version for loading and saving configuration. Specific configuration settings take different default values depending on the selected profile and version.

While the profile can be selected by user configuration and will remain over upgrades, frr will always write configurations using its current version. This means that, after upgrading, a `write file` may write out a slightly different configuration than what was read in.

Since the previous configuration is loaded with its version's defaults, but the new configuration is written with the new defaults, any default that changed between versions will result in an appropriate configuration entry being written out. **FRRouting configuration is sticky, staying consistent over upgrades.** Changed defaults will only affect new configuration.

Note that the loaded version persists into interactive configuration sessions. Commands executed in an interactive configuration session are no different from configuration loaded at startup. This means that when, say, you configure a new BGP peer, the defaults used for configuration are the ones selected by the last `frr version` command.

Warning: Saving the configuration does not bump the daemons forward to use the new version for their defaults, but restarting them will, since they will then apply the new `frr version` command that was written out. Manually execute the `frr version` command in `show running-config` to avoid this intermediate state.

This is visible in `show running-config`:

```
Current configuration:
!
! loaded from 6.0
frr version 6.1-dev
frr defaults traditional
!
```

If you save and then restart with this configuration, the old defaults will no longer apply. Similarly, you could execute `frr version 6.1-dev`, causing the new defaults to apply and the `loaded from 6.0` comment to disappear.

Profiles

frr provides configuration profiles to adapt its default settings to various usage scenarios. Currently, the following profiles are implemented:

- `traditional` - reflects defaults adhering mostly to IETF standards or common practices in wide-area internet routing.
- `datacenter` - reflects a single administrative domain with intradomain links using aggressive timers.

Your distribution/installation may pre-set a profile through the `-F` command line option on all daemons. All daemons must be configured for the same profile. The value specified on the command line is only a pre-set and any `frr defaults` statement in the configuration will take precedence.

Note: The profile must be the same across all daemons. Mismatches may result in undefined behavior.

You can freely switch between profiles without causing any interruption or configuration changes. All settings remain at their previous values, and `show running-configuration` output will have new output listing the previous default values as explicit configuration. New configuration, e.g. adding a BGP peer, will use the new defaults. To apply the new defaults for existing configuration, the previously-invisible old defaults that are now shown must be removed from the configuration.

Upgrade practices for interactive configuration

If you configure frr interactively and use the configuration writing functionality to make changes persistent, the following recommendations apply in regards to upgrades:

1. Skipping major versions should generally work but is still inadvisable. To avoid unneeded issue, upgrade one major version at a time and write out the configuration after each update.
2. After installing a new frr version, check the configuration for differences against your old configuration. If any defaults changed that affect your setup, lines may appear or disappear. If a new line appears, it was previously the default (or not supported) and is now necessary to retain previous behavior. If a line disappears, it previously wasn't the default, but now is, so it is no longer necessary.
3. Check the log files for deprecation warnings by using `grep -i deprecat`.
4. After completing each upgrade, save the configuration and either restart frr or execute `frr version <CURRENT>` to ensure defaults of the new version are fully applied.

Upgrade practices for autogenerated configuration

When using frr with generated configurations (e.g. Ansible, Puppet, etc.), upgrade considerations differ somewhat:

1. Always write out a `frr version` statement in the configurations you generate. This ensures that defaults are applied consistently.
2. Try to not run more distinct versions of frr than necessary. Each version may need to be checked individually. If running a mix of older and newer installations, use the oldest version for the `frr version` statement.
3. When rolling out upgrades, generate a configuration as usual with the old version identifier and load it. Check for any differences or deprecation warnings. If there are differences in the configuration, propagate these back to the configuration generator to minimize relying on actual default values.
4. After the last installation of an old version is removed, change the configuration generation to a newer `frr version` as appropriate. Perform the same checks as when rolling out upgrades.

2.1.2 Terminal Mode Commands

`write terminal`
Displays the current configuration to the vty interface.

`write file`
Write current configuration to configuration file.

`configure [terminal]`
Change to configuration mode. This command is the first step to configuration.

`terminal length (0-512)`
Set terminal display length to `(0-512)`. If length is 0, no display control is performed.

`who`
Show a list of currently connected vty sessions.

`list`
List all available commands.

show version
Show the current version of frr and its build host information.

show logging
Shows the current configuration of the logging system. This includes the status of all logging destinations.

show log-filter
Shows the current log filters applied to each daemon.

show memory [DAEMON]
Show information on how much memory is used for which specific things in frr. Output may vary depending on system capabilities but will generally look something like this:

```
frr# show memory
System allocator statistics:
  Total heap allocated:  1584 KiB
  Holding block headers: 0 bytes
  Used small blocks:     0 bytes
  Used ordinary blocks:  1484 KiB
  Free small blocks:     2096 bytes
  Free ordinary blocks:  100 KiB
  Ordinary blocks:       2
  Small blocks:          60
  Holding blocks:        0
(see system documentation for 'mallinfo' for meaning)
--- qmem libfrr ---
Buffer                        :          3      24                  72
Buffer data                   :          1    4120                4120
Host config                   :          3   (variably sized)       72
Command Tokens                :       3427      72              247160
Command Token Text            :       2555   (variably sized)    83720
Command Token Help            :       2555   (variably sized)    61720
Command Argument              :          2   (variably sized)       48
Command Argument Name         :        641   (variably sized)    15672
[...]
--- qmem Label Manager ---
--- qmem zebra ---
ZEBRA VRF                     :          1     912                 920
Route Entry                   :         11      80                 968
Static route                  :          1     192                 200
RIB destination               :          8      48                 448
RIB table info                :          4      16                  96
Nexthop tracking object       :          1     200                 200
Zebra Name Space              :          1     312                 312
--- qmem Table Manager ---
```

To understand system allocator statistics, refer to your system's `mallinfo(3)` man page.

Below these statistics, statistics on individual memory allocation types in frr (so-called *MTYPEs*) is printed:

- the first column of numbers is the current count of allocations made for the type (the number decreases when items are freed.)
- the second column is the size of each item. This is only available if allocations on a type are always made with the same size.
- the third column is the total amount of memory allocated for the particular type, including padding applied by malloc. This means that the number may be larger than the first column multiplied by the second.

Overhead incurred by malloc's bookkeeping is not included in this, and the column may be missing if system support is not available.

When executing this command from `vtysh`, each of the daemons' memory usage is printed sequentially. You can specify the daemon's name to print only its memory usage.

`show motd`
Show current motd banner.

`show history`
Dump the vtysh cli history.

`logmsg LEVEL MESSAGE`
Send a message to all logging destinations that are enabled for messages of the given severity.

`find REGEX...`
This command performs a regex search across all defined commands in all modes. As an example, suppose you're in enable mode and can't remember where the command to turn OSPF segment routing on is:

```
frr# find segment-routing on
  (ospf)  segment-routing on
  (isis)  segment-routing on
```

The CLI mode is displayed next to each command. In this example, `segment-routing on` is under the *router ospf* mode.

Similarly, suppose you want a listing of all commands that contain "l2vpn" and "neighbor":

```
frr# find l2vpn.*neighbor
  (view)  show [ip] bgp l2vpn evpn neighbors <A.B.C.D|X:X::X:X|WORD> advertised-
↪routes [json]
  (view)  show [ip] bgp l2vpn evpn neighbors <A.B.C.D|X:X::X:X|WORD> routes [json]
  (view)  show [ip] bgp l2vpn evpn rd ASN:NN_OR_IP-ADDRESS:NN neighbors <A.B.C.
↪D|X:X::X:X|WORD> advertised-routes [json]
  (view)  show [ip] bgp l2vpn evpn rd ASN:NN_OR_IP-ADDRESS:NN neighbors <A.B.C.
↪D|X:X::X:X|WORD> routes [json]
  ...
```

Note that when entering spaces as part of a regex specification, repeated spaces will be compressed into a single space for matching purposes. This is a consequence of spaces being used to delimit CLI tokens. If you need to match more than one space, use the `\s` escape.

POSIX Extended Regular Expressions are supported.

`show thread cpu [r|w|t|e|x]`
This command displays system run statistics for all the different event types. If no options is specified all different run types are displayed together. Additionally you can ask to look at (r)ead, (w)rite, (t)imer, (e)vent and e(x)ecute thread event types. If you have compiled with disable-cpu-time then this command will not show up.

`show thread poll`
This command displays FRR's poll data. It allows a glimpse into how we are setting each individual fd for the poll command at that point in time.

`show thread timers`
This command displays FRR's timer data for timers that will pop in the future.

`show yang operational-data XPATH [{format <json|xml>|translate TRANSLATOR|with-config}] DAEMON`
Display the YANG operational data starting from XPATH. The default format is JSON, but can be displayed in XML as well.

Normally YANG operational data are located inside containers marked as *read-only*.

Optionally it is also possible to display configuration leaves in addition to operational data with the option *with-config*. This option enables the display of configuration leaves with their currently configured value (if the leaf is optional it will only show if it was created or has a default value).

2.1.3 Common Invocation Options

These options apply to all frr daemons.

`-d, --daemon`
Run in daemon mode.

`-f, --config_file` `<file>`
Set configuration file name.

`-h, --help`
Display this help and exit.

`-i, --pid_file` `<file>`
Upon startup the process identifier of the daemon is written to a file, typically in `/var/run`. This file can be used by the init system to implement commands such as `.../init.d/zebra status`, `.../init.d/zebra restart` or `.../init.d/zebra stop`.

The file name is an run-time option rather than a configure-time option so that multiple routing daemons can be run simultaneously. This is useful when using frr to implement a routing looking glass. One machine can be used to collect differing routing views from differing points in the network.

`-A, --vty_addr` `<address>`
Set the VTY local address to bind to. If set, the VTY socket will only be bound to this address.

`-P, --vty_port` `<port>`
Set the VTY TCP port number. If set to 0 then the TCP VTY sockets will not be opened.

`-u` `<user>`
Set the user and group to run as.

`-N` `<namespace>`
Set the namespace that the daemon will run in. A "/<namespace>" will be added to all files that use the statedir. If you have "/var/run/frr" as the default statedir then it will become "/var/run/frr/<namespace>".

`-o, --vrfdefaultname` `<name>`
Set the name used for the *Default VRF* in CLI commands and YANG models. This option must be the same for all running daemons. By default, the name is "default".

See also:

Virtual Routing and Forwarding

`-v, --version`
Print program version.

`--command-log-always`
Cause the daemon to always log commands entered to the specified log file. This also makes the *no log commands* command dissallowed. Enabling this is suggested if you have need to track what the operator is doing on this router.

`--log` `<stdout|syslog|file:/path/to/log/file>`
When initializing the daemon, setup the log to go to either stdout, syslog or to a file. These values will be displayed as part of a show run. Additionally they can be overridden at runtime if desired via the normal log commands.

--log-level <emergencies|alerts|critical|errors|warnings|notifications|informational|debugging>
When initializing the daemon, allow the specification of a default log level at startup from one of the specified levels.

--tcli
Enable the transactional CLI mode.

--limit-fds <number>
Limit the number of file descriptors that will be used internally by the FRR daemons. By default, the daemons use the system ulimit value.

2.1.4 Loadable Module Support

FRR supports loading extension modules at startup. Loading, reloading or unloading modules at runtime is not supported (yet). To load a module, use the following command line option at daemon startup:

-M, --module <module:options>
Load the specified module, optionally passing options to it. If the module name contains a slash (/), it is assumed to be a full pathname to a file to be loaded. If it does not contain a slash, the /usr/lib/frr/modules directory is searched for a module of the given name; first with the daemon name prepended (e.g. `zebra_mod` for `mod`), then without the daemon name prepended.

This option is available on all daemons, though some daemons may not have any modules available to be loaded.

The SNMP Module

If SNMP is enabled during compile-time and installed as part of the package, the `snmp` module can be loaded for the *Zebra*, *bgpd*, *ospfd*, *ospf6d* and *ripd* daemons.

The module ignores any options passed to it. Refer to *SNMP Support* for information on its usage.

The FPM Module

If FPM is enabled during compile-time and installed as part of the package, the `fpm` module can be loaded for the *zebra* daemon. This provides the Forwarding Plane Manager ("FPM") API.

The module expects its argument to be either `Netlink` or `protobuf`, specifying the encapsulation to use. `Netlink` is the default, and `protobuf` may not be available if the module was built without protobuf support. Refer to *zebra FIB push interface* for more information.

2.1.5 Virtual Terminal Interfaces

VTY – Virtual Terminal [aka TeletYpe] Interface is a command line interface (CLI) for user interaction with the routing daemon.

VTY Overview

VTY stands for Virtual TeletYpe interface. It means you can connect to the daemon via the telnet protocol.

To enable a VTY interface, you have to setup a VTY password. If there is no VTY password, one cannot connect to the VTY interface at all.

```
% telnet localhost 2601
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.

Hello, this is |PACKAGE_NAME| (version |PACKAGE_VERSION|)
|COPYRIGHT_STR|

User Access Verification

Password: XXXXX
Router> ?
  enable .  .  .  Turn on privileged commands
  exit   .  .  .  Exit current mode and down to previous mode
  help   .  .  .  Description of the interactive help system
  list   .  .  .  Print command list
  show   .  .  .  Show system inform

  wh. . .  Display who is on a vty
Router> enable
Password: XXXXX
Router# configure terminal
Router(config)# interface eth0
Router(config-if)# ip address 10.0.0.1/8
Router(config-if)# ^Z
Router#
```

VTY Modes

There are three basic VTY modes:

There are commands that may be restricted to specific VTY modes.

VTY View Mode

This mode is for read-only access to the CLI. One may exit the mode by leaving the system, or by entering *enable* mode.

VTY Enable Mode

This mode is for read-write access to the CLI. One may exit the mode by leaving the system, or by escaping to view mode.

VTY Other Modes

This page is for describing other modes.

VTY CLI Commands

Commands that you may use at the command-line are described in the following three subsubsections.

CLI Movement Commands

These commands are used for moving the CLI cursor. The `C` character means press the Control Key.

`C-f` / **`LEFT`** Move forward one character.

`C-b` / **`RIGHT`** Move backward one character.

`M-f` Move forward one word.

`M-b` Move backward one word.

`C-a` Move to the beginning of the line.

`C-e` Move to the end of the line.

CLI Editing Commands

These commands are used for editing text on a line. The `C` character means press the Control Key.

`C-h` / **`DEL`** Delete the character before point.

`C-d` Delete the character after point.

`M-d` Forward kill word.

`C-w` Backward kill word.

`C-k` Kill to the end of the line.

`C-u` Kill line from the beginning, erasing input.

`C-t` Transpose character.

CLI Advanced Commands

There are several additional CLI commands for command line completions, insta-help, and VTY session management.

`C-c` Interrupt current input and moves to the next line.

`C-z` End current configuration session and move to top node.

`C-n` / `DOWN` Move down to next line in the history buffer.

`C-p` / `UP` Move up to previous line in the history buffer.

`TAB` Use command line completion by typing `TAB`.

`?` You can use command line help by typing `help` at the beginning of the line. Typing `?` at any point in the line will show possible completions.

Pipe Actions

VTY supports optional modifiers at the end of commands that perform postprocessing on command output or modify the action of commands. These do not show up in the `?` or `TAB` suggestion lists.

`... | include REGEX` Filters the output of the preceding command, including only lines which match the POSIX Extended Regular Expression `REGEX`. Do not put the regex in quotes.

Examples:

```
frr# show ip bgp sum json | include remoteAs
      "remoteAs":0,
      "remoteAs":455,
      "remoteAs":99,
```

```
frr# show run | include neigh.*[0-9]{2}\.0\.[2-4]\.[0-9]*
 neighbor 10.0.2.106 remote-as 99
 neighbor 10.0.2.107 remote-as 99
 neighbor 10.0.2.108 remote-as 99
 neighbor 10.0.2.109 remote-as 99
 neighbor 10.0.2.110 remote-as 99
 neighbor 10.0.3.111 remote-as 111
```

2.2 Extended Logging Target

After creating one or more extended logging targets with the *`log extended EXTLOGNAME`* command, the target(s) must be configured for the desired logging output.

Each extended log target supports emitting log messages in one of the following formats:

- `rfc5424` - **RFC 5424** - modern syslog with ISO 8601 timestamps, time zone and structured data (key/value pairs) support
- `rfc3164` - **RFC 3164** - legacy BSD syslog, timestamps with 1 second granularity
- `local-syslog` - same as **RFC 3164**, but without the hostname field
- `journald` - systemd's native journald protocol. This protocol also supports structured data (key/value pairs).

2.2.1 Destinations

The output location is configured with the following subcommands:

`destination none`
Disable the target while retaining its remaining configuration.

`destination syslog [supports-rfc5424]`
Send log messages to the system's standard log destination (`/dev/log`). This does not use the C library's `syslog()` function, instead writing directly to `/dev/log`.

On NetBSD and FreeBSD, the RFC5424 format is automatically used when the OS version is recent enough (5.0 for NetBSD, 12.0 for FreeBSD). Unfortunately, support for this format cannot be autodetected otherwise, and particularly on Linux systems must be enabled manually.

`destination journald`
Send log messages to systemd's journald.

`destination <stdout|stderr|fd <(0-63)|envvar WORD>> [format FORMAT]`
Send log messages to one of the daemon's file descriptors. The `fd (0-63)` and `fd envvar WORD` variants are intended to work with the shell's `command 3>something` and bash's `command {ENVVAR}>something` I/O redirection specifiers.

Only file descriptors open at a daemon's startup time can be used for this; accidental misuse of a file descriptor that has been opened by FRR itself is prevented.

Using FIFOs with this option will work but is unsupported and can cause daemons to hang or crash depending on reader behavior.

Format defaults to RFC5424 if none is specified.

Note: When starting FRR daemons from custom shell scripts, make sure not to leak / leave extraneous file descriptors open. FRR daemons do not close these.

`destination file PATH [create [{user WORD|group WORD|mode PERMS}]|no-create]`
Log to a regular file. File permissions can be specified when FRR creates the file itself.

Format defaults to RFC5424 if none is specified.

Note: FRR will never change permissions or ownership on an existing log file. In many cases, FRR will also not have permissions to set user and group arbitrarily.

`destination unix PATH [format FORMAT]`
Connect to a UNIX domain socket and send log messages there. This will autodetect `SOCK_STREAM`, `SOCK_SEQPACKET` and `SOCK_DGRAM` and adjust behavior appropriately.

2.2.2 Options

`priority PRIORITY`
Select minimum priority of messages to send to this target. Defaults to *debugging*.

`facility FACILITY`
Select syslog facility for messages on this target. Defaults to *daemon*. The `log facility [FACILITY]` command does not affect extended targets.

`timestamp precision (0-9)`
Set desired number of sub-second timestamp digits. This only has an effect for RFC5424 and journald format targets; the RFC3164 and local-syslogd formats do not support any sub-second digits.

`timestamp local-time`
Use the local system timezone for timestamps rather than UTC (the default.)

RFC5424 and journald formats include zone information (`Z` or `+-NN:NN` suffix in ISO8601). RFC3164 and local-syslogd offer no way of identifying the time zone used, care must be taken that this option and the receiver are configured identically, or the timestamp is replaced at the receiver.

Note: FRR includes a timestamp in journald messages, but journald always provides its own timestamp.

`structured-data <code-location|version|unique-id|error-category|format-args>`
Select additional key/value data to be included for the RFC5424 and journald formats. Refer to the next section for details.

`unique-id` and `error-category` are enabled by default.

Warning: Log messages can grow in size significantly when enabling additional data.

2.2.3 Structured data

When using the RFC5424 or journald formats, FRR can provide additional metadata for log messages as key/value pairs. The following information can be added in this way:

Switch	5424 group	5424 item(s)	journald field	Contents
always active	`location@50145`	`tid`	`TID`	Thread ID
always active	`location@50145`	`instance`	`FRR_INSTANCE`	Multi-instance number
`unique-id`	`location@50145`	`id`	`FRR_ID`	`XXXXX-XXXXX` unique message identifier
`error-category`	`location@50145`	`ec`	`FRR_EC`	Integer error category number
`code-location`	`location@50145`	`file`	`CODE_FILE`	Source code file name
`code-location`	`location@50145`	`line`	`CODE_LINE`	Source code line number
`code-location`	`location@50145`	`func`	`CODE_FUNC`	Source code function name
`format-args`	`args@50145`	`argN`	`FRR_ARGn`	Message printf format arguments (n = 1..16)
`version`	`origin`	multiple	n/a	FRR version information (IETF format)

The information added by `version` is `[origin enterpriseId="50145" software="FRRouting" swVersion="..."]` and is the same for all log messages. (Hence makes little sense to include in most scenarios.) 50145 is the FRRouting IANA Enterprise Number.

Crashlogs / backtraces do not include any additional information since it cannot safely be retrieved from a crash handler. However, all of the above destinations will deliver crashlogs.

2.2.4 Restart and Reconfiguration caveats

FRR uses "add-delete" semantics when reconfiguring log targets of any type (including both extended targets mentioned here as well as the global *`log stdout LEVEL`* and *`log syslog [LEVEL]`* variants.) This means that when changing logging configuration, log messages from threads executing in parallel may be duplicated for a brief window of time.

For the `unix`, `syslog` and `journald` extended destinations, messages can be lost when the receiver is restarted without the use of socket activation (i.e. keeping the receiver socket open.) FRR does not buffer log messages for later delivery, meaning anything logged while the receiver is unavailable is lost. Since systemd provides socket activation for journald, no messages will be lost on the `journald` target.

2.3 VTY shell

vtysh provides a combined frontend to all FRR daemons in a single combined session. It is enabled by default at build time, but can be disabled through the *`--disable-vtysh`* option to the configure script.

vtysh has a configuration file, `vtysh.conf`. The location of that file cannot be changed from /etc/frr since it contains options controlling authentication behavior. This file will also not be written by configuration-save commands, it is intended to be updated manually by an administrator with an external editor.

Warning: This also means the `hostname` and `banner motd` commands (which both do have effect for vtysh) need to be manually updated in `vtysh.conf`.

`copy FILENAME running-config`
Process and load a configuration file manually; each line in the file is read and processed as if it were being typed (or piped) to vtysh.

2.3.1 Live logs

`terminal monitor [DAEMON]`
Receive and display log messages.

It is not currently possible to change the minimum message priority (fixed to debug) or output formatting. These will likely be made configurable in the future.

Log messages are received asynchronously and may be printed both during command execution as well as while on the prompt. They are printed to stderr, unlike regular CLI output which is printed to stdout. The intent is that stdin/stdout might be driven by some script while log messages are visible on stderr. If stdout and stderr are the same file, the prompt and pending input will be cleared and reprinted appropriately.

Note: If `vtysh` cannot keep up, some log messages may be lost. The daemons do **not** wait for, get blocked by, or buffer messages for `vtysh`.

2.3.2 Pager usage

vtysh can call an external paging program (e.g. *more* or *less*) to paginate long output from commands. This feature used to be enabled by default but is now controlled by the `VTYSH_PAGER` environment variable and the `terminal paginate` command:

`VTYSH_PAGER`

If set, the `VTYSH_PAGER` environment variable causes *vtysh* to pipe output from commands through the given command. Note that this happens regardless of the length of the output. As such, standard pager behavior (particularly waiting at the end of output) tends to be annoying to the user. Using `less -EFX` is recommended for a better user experience.

If this environment variable is unset, *vtysh* defaults to not using any pager.

This variable should be set by the user according to their preferences, in their `~/.profile` file.

`terminal paginate`

Enables/disables vtysh output pagination. This command is intended to be placed in `vtysh.conf` to set a system-wide default. If this is enabled but `VTYSH_PAGER` is not set, the system default pager (likely `more` or `/usr/bin/pager`) will be used.

2.3.3 Permissions and setup requirements

vtysh connects to running daemons through Unix sockets located in /var/run/frr. Running vtysh thus requires access to that directory, plus membership in the frrvty group (which is the group that the daemons will change ownership of their sockets to).

To restrict access to FRR configuration, make sure no unauthorized users are members of the frrvty group.

Warning: VTYSH implements a CLI option `-u, --user` that disallows entering the characters "en" on the command line, which ideally restricts access to configuration commands. However, VTYSH was never designed to be a privilege broker and is not built using secure coding practices. No guarantees of security are provided for this option and under no circumstances should this option be used to provide any semblance of security or read-only access to FRR.

PAM support (experimental)

vtysh has working (but rather useless) PAM support. It will perform an "authenticate" PAM call using frr as service name. No other (accounting, session, password change) calls will be performed by vtysh.

Users using vtysh still need to have appropriate access to the daemons' VTY sockets, usually by being member of the frrvty group. If they have this membership, PAM support is useless since they can connect to daemons and issue commands using some other tool. Alternatively, the *vtysh* binary could be made SGID (set group ID) to the frrvty group.

Warning: No security guarantees are made for this configuration.

`username USERNAME nopassword`

If PAM support is enabled at build-time, this command allows disabling the use of PAM on a per-user basis. If vtysh finds that an user is trying to use vtysh and a "nopassword" entry is found, no calls to PAM will be made at all.

2.3.4 Integrated configuration mode

Integrated configuration mode uses a single configuration file, `frr.conf`, for all daemons. This replaces the individual files like `zebra.conf` or `bgpd.conf`.

`frr.conf` is located in /etc/frr. All daemons check for the existence of this file at startup, and if it exists will not load their individual configuration files. Instead, `vtysh -b` must be invoked to process `frr.conf` and apply its settings to the individual daemons.

Warning: *vtysh -b* must also be executed after restarting any daemon.

Configuration saving, file ownership and permissions

The `frr.conf` file is not written by any of the daemons; instead *vtysh* contains the necessary logic to collect configuration from all of the daemons, combine it and write it out.

Warning: Daemons must be running for *vtysh* to be able to collect their configuration. Any configuration from non-running daemons is permanently lost after doing a configuration save.

Since the *vtysh* command may be running as ordinary user on the system, configuration writes will be tried through *watchfrr*, using the `write integrated` command internally. Since *watchfrr* is running as superuser, *vtysh* is able to ensure correct ownership and permissions on `frr.conf`.

If *watchfrr* is not running or the configuration write fails, *vtysh* will attempt to directly write to the file. This is likely to fail if running as unprivileged user; alternatively it may leave the file with incorrect owner or permissions.

Writing the configuration can be triggered directly by invoking *vtysh -w*. This may be useful for scripting. Note this command should be run as either the superuser or the FRR user.

We recommend you do not mix the use of the two types of files.

`service integrated-vtysh-config`

Control whether integrated `frr.conf` file is written when 'write file' is issued.

These commands need to be placed in `vtysh.conf` to have any effect. Note that since `vtysh.conf` is not written by FRR itself, they therefore need to be manually placed in that file.

This command has 3 states:

service integrated-vtysh-config *vtysh* will always write `frr.conf`.

no service integrated-vtysh-config *vtysh* will never write `frr.conf`; instead it will ask daemons to write their individual configuration files.

Neither option present (default) *vtysh* will check whether `frr.conf` exists. If it does, configuration writes will update that file. Otherwise, writes are performed through the individual daemons.

This command is primarily intended for packaging/distribution purposes, to preset one of the two operating modes and ensure consistent operation across installations.

`write integrated`

Unconditionally (regardless of `service integrated-vtysh-config` setting) write out integrated `frr.conf` file through *watchfrr*. If *watchfrr* is not running, this command is unavailable.

Warning: Configuration changes made while some daemon is not running will be invisible to that daemon. The daemon will start up with its saved configuration (either in its individual configuration file, or in `frr.conf`). This is particularly troublesome for route-maps and prefix lists, which would otherwise be synchronized between daemons.

2.4 Northbound gRPC

gRPC provides a combined front end to all FRR daemons using the YANG northbound. It is currently disabled by default due its experimental stage, but it can be enabled with *`--enable-grpc`* option in the configure script.

2.4.1 Northbound gRPC Features

- Get/set configuration using JSON/XML/XPath encondings.
- Execute YANG RPC calls.
- Lock/unlock configuration.
- Create/edit/load/update/commit candidate configuration.
- List/get transactions.

Note: There is currently no support for YANG notifications.

Note: You can find more information on how to code programs to interact with FRR by reading the gRPC Programming Language Bindings section in the developer's documentation.

2.4.2 Daemon gRPC Configuration

The *gRPC* module accepts the following run time option:

- `port`: the port to listen to (defaults to `50051`).

Note: At the moment only localhost connections with no SSL/TLS are supported.

To configure FRR daemons to listen to gRPC you need to append the following parameter to the daemon's command line: `-M grpc` (optionally `-M grpc:PORT` to specify listening port).

To do that in production you need to edit the `/etc/frr/daemons` file so the daemons get started with the command line argument. Example:

```
# other daemons...
bfdd_options="  --daemon -A 127.0.0.1 -M grpc"
```

2.5 Filtering

FRR provides many very flexible filtering features. Filtering is used for both input and output of the routing information. Once filtering is defined, it can be applied in any direction.

2.5.1 IP Access List

access-list NAME [seq (1-4294967295)] permit IPV4-NETWORK

access-list NAME [seq (1-4294967295)] deny IPV4-NETWORK

seq seq *number* can be set either automatically or manually. In the case that sequential numbers are set manually, the user may pick any number less than 4294967295. In the case that sequential number are set automatically, the sequential number will increase by a unit of five (5) per list. If a list with no specified sequential number is created after a list with a specified sequential number, the list will automatically pick the next multiple of five (5) as the list number. For example, if a list with number 2 already exists and a new list with no specified number is created, the next list will be numbered 5. If lists 2 and 7 already exist and a new list with no specified number is created, the new list will be numbered 10.

Basic filtering is done by *access-list* as shown in the following example.

```
access-list filter deny 10.0.0.0/9
access-list filter permit 10.0.0.0/8
access-list filter seq 13 permit 10.0.0.0/7
```

show <ip|ipv6> access-list [json]
Display all IPv4 or IPv6 access lists.

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

show <ip|ipv6> access-list WORD [json]
Display the specified IPv4 or IPv6 access list.

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

2.5.2 IP Prefix List

ip prefix-list provides the most powerful prefix based filtering mechanism. In addition to *access-list* functionality, *ip prefix-list* has prefix length range specification and sequential number specification. You can add or delete prefix based filters to arbitrary points of prefix-list using sequential number specification.

If no ip prefix-list is specified, it acts as permit. If *ip prefix-list* is defined, and no match is found, default deny is applied.

ip prefix-list NAME (permit|deny) PREFIX [le LEN] [ge LEN]

ip prefix-list NAME seq NUMBER (permit|deny) PREFIX [le LEN] [ge LEN]
You can create *ip prefix-list* using above commands.

seq seq *number* can be set either automatically or manually. In the case that sequential numbers are set manually, the user may pick any number less than 4294967295. In the case that sequential number are set automatically, the sequential number will increase by a unit of five (5) per list. If a list with no specified sequential number is created after a list with a specified sequential number, the list will automatically pick the next multiple of five (5) as the list number. For example, if a list with number 2 already exists and a new list with no specified number is created, the next list will be numbered 5. If lists 2 and 7 already exist and a new list with no specified number is created, the new list will be numbered 10.

le Specifies prefix length. The prefix list will be applied if the prefix length is less than or equal to the le prefix length.

ge Specifies prefix length. The prefix list will be applied if the prefix length is greater than or equal to the ge prefix length.

Less than or equal to prefix numbers and greater than or equal to prefix numbers can be used together. The order of the le and ge commands does not matter.

If a prefix list with a different sequential number but with the exact same rules as a previous list is created, an error will result. However, in the case that the sequential number and the rules are exactly similar, no error will result.

If a list with the same sequential number as a previous list is created, the new list will overwrite the old list.

Matching of IP Prefix is performed from the smaller sequential number to the larger. The matching will stop once any rule has been applied.

In the case of no le or ge command, the prefix length must match exactly the length specified in the prefix list.

ip prefix-list description

`ip prefix-list NAME description DESC`
Descriptions may be added to prefix lists. This command adds a description to the prefix list.

Showing ip prefix-list

`show ip prefix-list [json]`
Display all IP prefix lists.

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

`show ip prefix-list NAME [json]`
Show IP prefix list can be used with a prefix list name.

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

`show ip prefix-list NAME seq NUM [json]`
Show IP prefix list can be used with a prefix list name and sequential number.

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

`show ip prefix-list NAME A.B.C.D/M`
If the command longer is used, all prefix lists with prefix lengths equal to or longer than the specified length will be displayed. If the command first match is used, the first prefix length match will be displayed.

`show ip prefix-list NAME A.B.C.D/M longer`

`show ip prefix-list NAME A.B.C.D/M first-match`

`show ip prefix-list summary [json]`

`show ip prefix-list summary NAME [json]`

`show ip prefix-list detail [json]`

`show ip prefix-list detail NAME [json]`

`debug prefix-list NAME match <A.B.C.D/M|X:X::X:X/M> [address-mode]`
Execute the prefix list matching code for the specified list and prefix. Shows which entry matched, if any. (`address-mode` is used for PIM RP lookups and skips prefix length checks.)

The return value from this command is success only if the prefix-list result is to permit the prefix, so the command can be used in scripting.

Clear counter of ip prefix-list

`clear ip prefix-list [NAME [A.B.C.D/M]]`

Clears the counters of all IP prefix lists. Clear IP Prefix List can be used with a specified NAME or NAME and prefix.

2.6 Route Maps

Route maps provide a means to both filter and/or apply actions to route, hence allowing policy to be applied to routes.

For a route reflector to apply a `route-map` to reflected routes, be sure to include `bgp route-reflector allow-outbound-policy` in `router bgp` mode.

Route maps are an ordered list of route map entries. Each entry may specify up to four distinct sets of clauses:

Matching Conditions A route-map entry may, optionally, specify one or more conditions which must be matched if the entry is to be considered further, as governed by the Match Policy. If a route-map entry does not explicitly specify any matching conditions, then it always matches.

Set Actions A route-map entry may, optionally, specify one or more Set Actions to set or modify attributes of the route.

Matching Policy This specifies the policy implied if the *Matching Conditions* are met or not met, and which actions of the route-map are to be taken, if any. The two possibilities are:

- *permit*: If the entry matches, then carry out the *Set Actions*. Then finish processing the route-map, permitting the route, unless an *Exit Policy* action indicates otherwise.
- *deny*: If the entry matches, then finish processing the route-map and deny the route (return *deny*).

The *Matching Policy* is specified as part of the command which defines the ordered entry in the route-map. See below.

Call Action Call to another route-map, after any *Set Actions* have been carried out. If the route-map called returns *deny* then processing of the route-map finishes and the route is denied, regardless of the *Matching Policy* or the *Exit Policy*. If the called route-map returns *permit*, then *Matching Policy* and *Exit Policy* govern further behaviour, as normal.

Exit Policy An entry may, optionally, specify an alternative *Exit Policy* to take if the entry matched, rather than the normal policy of exiting the route-map and permitting the route. The two possibilities are:

- *next*: Continue on with processing of the route-map entries.
- *goto N*: Jump ahead to the first route-map entry whose order in the route-map is >= N. Jumping to a previous entry is not permitted.

The default action of a route-map, if no entries match, is to deny. I.e. a route-map essentially has as its last entry an empty *deny* entry, which matches all routes. To change this behaviour, one must specify an empty *permit* entry as the last entry in the route-map.

To summarise the above:

	Match	No Match
Permit	action	cont
Deny	deny	cont

action

- Apply *set* statements
- If *call* is present, call given route-map. If that returns a `deny`, finish processing and return `deny`.
- If *Exit Policy* is *next*, goto next route-map entry
- If *Exit Policy* is *goto*, goto first entry whose order in the list is >= the given order.
- Finish processing the route-map and permit the route.

deny The route is denied by the route-map (return `deny`).

cont goto next route-map entry

`show route-map [WORD] [json]`

Display data about each daemons knowledge of individual route-maps. If WORD is supplied narrow choice to that particular route-map.

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

`clear route-map counter [WORD]`

Clear counters that are being stored about the route-map utilization so that subsuquent show commands will indicate since the last clear. If WORD is specified clear just that particular route-map's counters.

2.6.1 Route Map Command

`route-map ROUTE-MAP-NAME (permit|deny) ORDER`

Configure the *order*'th entry in *route-map-name* with `Match Policy` of either *permit* or *deny*.

2.6.2 Route Map Match Command

`match ip address ACCESS_LIST`

Matches the specified *access_list*

`match ip address prefix-list PREFIX_LIST`

Matches the specified *PREFIX_LIST*

`match ip address prefix-len 0-32`

Matches the specified *prefix-len*. This is a Zebra specific command.

`match ipv6 address ACCESS_LIST`

Matches the specified *access_list*

`match ipv6 address prefix-list PREFIX_LIST`

Matches the specified *PREFIX_LIST*

`match ipv6 address prefix-len 0-128`

Matches the specified *prefix-len*. This is a Zebra specific command.

`match ip next-hop ACCESS_LIST`

Match the next-hop according to the given access-list.

`match ip next-hop address IPV4_ADDR`

This is a BGP specific match command. Matches the specified *ipv4_addr*.

`match ip next-hop prefix-list PREFIX_LIST`

Match the next-hop according to the given prefix-list.

`match ipv6 next-hop ACCESS_LIST`
Match the next-hop according to the given access-list.

`match ipv6 next-hop address IPV6_ADDR`
This is a BGP specific match command. Matches the specified *ipv6_addr*.

`match ipv6 next-hop prefix-list PREFIX_LIST`
Match the next-hop according to the given prefix-list.

`match as-path AS_PATH`
Matches the specified *as_path*.

`match metric METRIC`
Matches the specified *metric*.

`match tag TAG`
Matches the specified tag value associated with the route. This tag value can be in the range of (1-4294967295).

`match local-preference METRIC`
Matches the specified *local-preference*.

`match community COMMUNITY_LIST`
Matches the specified *community_list*

`match peer IPV4_ADDR`
This is a BGP specific match command. Matches the peer ip address if the neighbor was specified in this manner.

`match peer IPV6_ADDR`
This is a BGP specific match command. Matches the peer ipv6 address if the neighbor was specified in this manner.

`match peer INTERFACE_NAME`
This is a BGP specific match command. Matches the peer interface name specified if the neighbor was specified in this manner.

`match peer PEER_GROUP_NAME`
This is a BGP specific match command. Matches the peer group name specified for the peer in question.

`match source-protocol PROTOCOL_NAME`
This is a ZEBRA and BGP specific match command. Matches the originating protocol specified.

`match source-instance NUMBER`
This is a ZEBRA specific match command. The number is a range from (0-255). Matches the originating protocols instance specified.

`match evpn route-type ROUTE_TYPE_NAME`
This is a BGP EVPN specific match command. It matches to EVPN route-type from type-1 (EAD route-type) to type-5 (Prefix route-type). User can provide in an integral form (1-5) or string form of route-type (i.e ead, macip, multicast, es, prefix).

`match evpn vni NUMBER`
This is a BGP EVPN specific match command which matches to EVPN VNI id. The number is a range from (1-6777215).

2.6.3 Route Map Set Command

`set tag TAG`
Set a tag on the matched route. This tag value can be from (1-4294967295). Additionally if you have compiled with the *`--enable-realms`* configure option. Tag values from (1-255) are sent to the Linux kernel as a realm value. Then route policy can be applied. See the tc man page. As a note realms cannot currently be used with the installation of nexthops as nexthop groups in the linux kernel.

`set ip next-hop IPV4_ADDRESS`
Set the BGP nexthop address to the specified IPV4_ADDRESS. For both incoming and outgoing route-maps.

`set ip next-hop peer-address`
Set the BGP nexthop address to the address of the peer. For an incoming route-map this means the ip address of our peer is used. For an outgoing route-map this means the ip address of our self is used to establish the peering with our neighbor.

`set ip next-hop unchanged`
Set the route-map as unchanged. Pass the route-map through without changing it's value.

`set ipv6 next-hop peer-address`
Set the BGP nexthop address to the address of the peer. For an incoming route-map this means the ipv6 address of our peer is used. For an outgoing route-map this means the ip address of our self is used to establish the peering with our neighbor.

`set ipv6 next-hop prefer-global`
For Incoming and Import Route-maps if we receive a v6 global and v6 LL address for the route, then prefer to use the global address as the nexthop.

`set ipv6 next-hop global IPV6_ADDRESS`
Set the next-hop to the specified IPV6_ADDRESS for both incoming and outgoing route-maps.

`set local-preference LOCAL_PREF`
Set the BGP local preference to *local_pref*.

`set local-preference +LOCAL_PREF`
Add the BGP local preference to an existing *local_pref*.

`set local-preference -LOCAL_PREF`
Subtract the BGP local preference from an existing *local_pref*.

`set distance (1-255)`
Set the Administrative distance to use for the route. This is only locally significant and will not be dispersed to peers.

`set weight WEIGHT`
Set the route's weight.

`set metric <[+|-](1-4294967295)|rtt|+rtt|-rtt>`
Set the route metric. When used with BGP, set the BGP attribute MED to a specific value. Use +/- to add or subtract the specified value to/from the existing/MED. Use *rtt* to set the MED to the round trip time or *+rtt*/*-rtt* to add/subtract the round trip time to/from the MED.

`set min-metric <(0-4294967295)>`
Set the minimum meric for the route.

`set max-metric <(0-4294967295)>`
Set the maximum meric for the route.

`set aigp-metric <igp-metric|(1-4294967295)>`
Set the BGP attribute AIGP to a specific value. If `igp-metric` is specified, then the value is taken from the IGP protocol, otherwise an arbitrary value.

set as-path prepend AS_PATH
Set the BGP AS path to prepend.

set as-path exclude AS-NUMBER...
Drop AS-NUMBER from the BGP AS path.

set community COMMUNITY
Set the BGP community attribute.

set ipv6 next-hop local IPV6_ADDRESS
Set the BGP-4+ link local IPv6 nexthop address.

set origin ORIGIN <egp|igp|incomplete>
Set BGP route origin.

set table (1-4294967295)
Set the BGP table to a given table identifier

set sr-te color (1-4294967295)
Set the color of a SR-TE Policy to be applied to a learned route. The SR-TE Policy is uniquely determined by the color and the BGP nexthop.

set l3vpn next-hop encapsulation gre
Accept L3VPN traffic over GRE encapsulation.

2.6.4 Route Map Call Command

call NAME
Call route-map *name*. If it returns deny, deny the route and finish processing the route-map.

2.6.5 Route Map Exit Action Command

on-match next

continue
Proceed on to the next entry in the route-map.

on-match goto N

continue N
Proceed processing the route-map at the first entry whose order is >= N

2.6.6 Route Map Optimization Command

route-map ROUTE-MAP-NAME optimization
Enable route-map processing optimization for *route-map-name*. The optimization is enabled by default. Instead of sequentially passing through all the route-map indexes until a match is found, the search for the best-match index will be based on a look-up in a prefix-tree. A per-route-map prefix-tree will be constructed for this purpose. The prefix-tree will compose of all the prefixes in all the prefix-lists that are included in the match rule of all the sequences of a route-map.

2.6.7 Route Map Examples

A simple example of a route-map:

```
route-map test permit 10
 match ip address 10
 set local-preference 200
```

This means that if a route matches ip access-list number 10 it's local-preference value is set to 200.

See *Miscellaneous Configuration Examples* for examples of more sophisticated usage of route-maps, including of the `call` action.

2.7 Affinity Maps

Affinity maps provide a means of configuring Standard Admininistrative-Group (RFC3630, RFC5305 and RFC5329) and Extended Admininistrative-Group (RFC7308). An affinity-map maps a specific bit position to a human readable-name.

An affinity refers to a color or a ressource class in the Traffic Engineering terminology. The bit position means the position of the bit set starting from the least significant bit. For example, if the affinity 'blue' has bit position 0 the extended Admin-Group value will be 0x01. If the affinity 'red' bit position 2 was added to a link in combination with the 'blue' affinity, the Admin-Group value would be 0x05.

2.7.1 Command

`affinity-map NAME bit-position (0-1023)`

Map the affinity name NAME to the bit-position. The bit-position is the key so that only one name can be mapped to particular bit-position.

`no affinity-map NAME`

Remove the affinity-map mapping.

Affinity-maps with a bit-position value higher than 31 are not compatible with Standard Admininistrative-Group. The CLI disallow the usage of such affinity-maps when Standard Admininistrative-Groups are required.

2.8 IPv6 Support

FRR fully supports IPv6 routing. As described so far, FRR supports RIPng, OSPFv3, and BGP-4+. You can give IPv6 addresses to an interface and configure static IPv6 routing information. FRR IPv6 also provides automatic address configuration via a feature called `address auto configuration`. To do it, the router must send router advertisement messages to the all nodes that exist on the network.

Previous versions of FRR could be built without IPv6 support. This is no longer possible.

2.8.1 Router Advertisement

show ipv6 nd ra-interfaces [vrf <VRFNAME|all>]
Show configured route advertisement interfaces. VRF subcommand only applicable for netns-based vrfs.

ipv6 nd suppress-ra
Don't send router advertisement messages. The no form of this command enables sending RA messages.

ipv6 nd prefix ipv6prefix [valid-lifetime] [preferred-lifetime] [off-link] [no-autoconfig] [router-addr
Configuring the IPv6 prefix to include in router advertisements. Several prefix specific optional parameters and flags may follow:

- `valid-lifetime`: the length of time in seconds during what the prefix is valid for the purpose of on-link determination. Value `infinite` represents infinity (i.e. a value of all one bits (`0xffffffff`)). Range: `(0-4294967295)` Default: `2592000`
- `preferred-lifetime`: the length of time in seconds during what addresses generated from the prefix remain preferred. Value `infinite` represents infinity. Range: `(0-4294967295)` Default: `604800`
- `off-link`: indicates that advertisement makes no statement about on-link or off-link properties of the prefix. Default: not set, i.e. this prefix can be used for on-link determination.
- `no-autoconfig`: indicates to hosts on the local link that the specified prefix cannot be used for IPv6 autoconfiguration.

  Default: not set, i.e. prefix can be used for autoconfiguration.
- `router-address`: indicates to hosts on the local link that the specified prefix contains a complete IP address by setting R flag.

  Default: not set, i.e. hosts do not assume a complete IP address is placed.

ipv6 nd ra-interval [(1-1800)]
The maximum time allowed between sending unsolicited multicast router advertisements from the interface, in seconds. Default: `600`

ipv6 nd ra-interval [msec (70-1800000)]
The maximum time allowed between sending unsolicited multicast router advertisements from the interface, in milliseconds. Default: `600000`

ipv6 nd ra-fast-retrans
RFC4861 states that consecutive RA packets should be sent no more frequently than three seconds apart. FRR by default allows faster transmissions of RA packets in order to speed convergence and neighbor establishment, particularly for unnumbered peering. By turning off ipv6 nd ra-fast-retrans, the implementation is compliant with the RFC at the cost of slower convergence and neighbor establishment. Default: enabled

ipv6 nd ra-retrans-interval [(0-4294967295)]
The value to be placed in the retrans timer field of router advertisements sent from the interface, in msec. Indicates the interval between router advertisement retransmissions. Setting the value to zero indicates that the value is unspecified by this router. Must be between zero or 4294967295 msec. Default: `0`

ipv6 nd ra-hop-limit [(0-255)]
The value to be placed in the hop count field of router advertisements sent from the interface, in hops. Indicates the maximum diameter of the network. Setting the value to zero indicates that the value is unspecified by this router. Must be between zero or 255 hops. Default: `64`

ipv6 nd ra-lifetime [(0-9000)]
The value to be placed in the Router Lifetime field of router advertisements sent from the interface, in seconds. Indicates the usefulness of the router as a default router on this interface. Setting the value to zero indicates that the router should not be considered a default router on this interface. Must be either zero or between value specified with `ipv6 nd ra-interval` (or default) and 9000 seconds. Default: `1800`

`ipv6 nd reachable-time [(1-3600000)]`
The value to be placed in the Reachable Time field in the Router Advertisement messages sent by the router, in milliseconds. The configured time enables the router to detect unavailable neighbors. The value zero means unspecified (by this router). Default: `0`

`ipv6 nd managed-config-flag`
Set/unset flag in IPv6 router advertisements which indicates to hosts that they should use managed (stateful) protocol for addresses autoconfiguration in addition to any addresses autoconfigured using stateless address autoconfiguration. Default: not set

`ipv6 nd other-config-flag`
Set/unset flag in IPv6 router advertisements which indicates to hosts that they should use administered (stateful) protocol to obtain autoconfiguration information other than addresses. Default: not set

`ipv6 nd home-agent-config-flag`
Set/unset flag in IPv6 router advertisements which indicates to hosts that the router acts as a Home Agent and includes a Home Agent Option. Default: not set

`ipv6 nd home-agent-preference [(0-65535)]`
The value to be placed in Home Agent Option, when Home Agent config flag is set, which indicates to hosts Home Agent preference. The default value of 0 stands for the lowest preference possible. Default: `0`

`ipv6 nd home-agent-lifetime [(0-65520)]`
The value to be placed in Home Agent Option, when Home Agent config flag is set, which indicates to hosts Home Agent Lifetime. The default value of 0 means to place the current Router Lifetime value.

Default: `0`

`ipv6 nd adv-interval-option`
Include an Advertisement Interval option which indicates to hosts the maximum time, in milliseconds, between successive unsolicited Router Advertisements. Default: not set

`ipv6 nd router-preference [(high|medium|low)]`
Set default router preference in IPv6 router advertisements per RFC4191. Default: medium

`ipv6 nd mtu [(1-65535)]`
Include an MTU (type 5) option in each RA packet to assist the attached hosts in proper interface configuration. The announced value is not verified to be consistent with router interface MTU.

Default: don't advertise any MTU option.

`ipv6 nd rdnss ipv6address [lifetime]`
Recursive DNS server address to advertise using the RDNSS (type 25) option described in RFC8106. Can be specified more than once to advertise multiple addresses. Note that hosts may choose to limit the number of RDNSS addresses to track.

Optional parameter:

- `lifetime`: the maximum time in seconds over which the specified address may be used for domain name resolution. Value `infinite` represents infinity (i.e. a value of all one bits (`0xffffffff`)). A value of 0 indicates that the address must no longer be used. Range: `(0-4294967295)` Default: `3 * ra-interval`

Default: do not emit RDNSS option

`ipv6 nd dnssl domain-name-suffix [lifetime]`
Advertise DNS search list using the DNSSL (type 31) option described in RFC8106. Specify more than once to advertise multiple domain name suffixes. Host implementations may limit the number of honored search list entries.

Optional parameter:

  - `lifetime`: the maximum time in seconds over which the specified domain suffix may be used in the course of name resolution. Value `infinite` represents infinity (i.e. a value of all one bits (`0xffffffff`)). A value of 0 indicates that the name suffix must no longer be used. Range: `(0-4294967295)` Default: `3 * ra-interval`

Default: do not emit DNSSL option

2.8.2 Router Advertisement Configuration Example

A small example:

```
interface eth0
 no ipv6 nd suppress-ra
 ipv6 nd prefix 2001:0DB8:5009::/64
```

See also:

- **RFC 2462** (IPv6 Stateless Address Autoconfiguration)
- **RFC 4861** (Neighbor Discovery for IP Version 6 (IPv6))
- **RFC 6275** (Mobility Support in IPv6)
- **RFC 4191** (Default Router Preferences and More-Specific Routes)
- **RFC 8106** (IPv6 Router Advertisement Options for DNS Configuration)

2.9 Kernel Interface

There are several different methods for reading kernel routing table information, updating kernel routing tables, and for looking up interfaces. FRR relies heavily on the Netlink (`man 7 netlink`) interface to communicate with the Kernel. However, other interfaces are still used in some parts of the code.

- **ioctl** This method is a very traditional way for reading or writing kernel information. *ioctl* can be used for looking up interfaces and for modifying interface addresses, flags, mtu settings and other types of information. Also, *ioctl* can insert and delete kernel routing table entries. It will soon be available on almost any platform which zebra supports, but it is a little bit ugly thus far, so if a better method is supported by the kernel, zebra will use that.
- **sysctl** This is a program that can lookup kernel information using MIB (Management Information Base) syntax. Normally, it only provides a way of getting information from the kernel. So one would usually want to change kernel information using another method such as *ioctl*.
- **proc filesystem** This is a special filesystem mount that provides an easy way of getting kernel information.
- **routing socket / Netlink** Netlink first appear in Linux kernel 2.0. It makes asynchronous communication between the kernel and FRR possible, similar to a routing socket on BSD systems. Netlink communication is done by reading/writing over Netlink socket.

2.10 SNMP Support

SNMP (Simple Network Managing Protocol) is a widely implemented feature for collecting network information from router and/or host. FRR itself does not support SNMP agent (server daemon) functionality but is able to connect to a SNMP agent using the the AgentX protocol (**RFC 2741**) and make the routing protocol MIBs available through it.

Note that SNMP Support needs to be enabled at compile-time and loaded as module on daemon startup. Refer to *Loadable Module Support* on the latter. If you do not start the daemons with snmp module support snmp will not work properly.

2.10.1 Getting and installing an SNMP agent

The supported SNMP agent is AgentX. We recommend to use the latest version of *net-snmp* which was formerly known as *ucd-snmp*. It is free and open software and available at http://www.net-snmp.org/ and as binary package for most Linux distributions.

2.10.2 NET-SNMP configuration

Routers with a heavy amount of routes (e.g. BGP full table) might experience problems with a hanging vtysh from time to time, 100% CPU on the snmpd or even crashes of the frr daemon(s) due to stalls within AgentX. Once snmp agents connects they start receiving a heavy amount of SNMP data (all the routes) which cannot be handled quick enough. It's recommended (by several vendors as well) to exclude these OID's unless you really need them, which can be achieved by amending the default view from SNMP

`/etc/snmp/snmpd.conf`:

```
# This is the default view
view all    included  .1 80
# Remove ipRouteTable from view
view all    excluded  .1.3.6.1.2.1.4.21
# Remove ipNetToMediaTable from view
view all    excluded  .1.3.6.1.2.1.4.22
# Remove ipNetToPhysicalPhysAddress from view
view all    excluded  .1.3.6.1.2.1.4.35
# Remove ipCidrRouteTable  from view
view all    excluded  .1.3.6.1.2.1.4.24
# Optionally protect SNMP private/secret values
view all    excluded  .1.3.6.1.6.3.15
view all    excluded  .1.3.6.1.6.3.16
view all    excluded  .1.3.6.1.6.3.18
# Optionally allow SNMP public info (sysName, location, etc)
view system included  .iso.org.dod.internet.mgmt.mib-2.system
```

2.10.3 AgentX configuration

To enable AgentX protocol support, FRR must have been build with the `--enable-snmp` or *–enable-snmp=agentx* option. Both the master SNMP agent (snmpd) and each of the FRR daemons must be configured. In `/etc/snmp/snmpd.conf`, the `master agentx` directive should be added. In each of the FRR daemons, `agentx` command will enable AgentX support.

`/etc/snmp/zebra.conf`:

```
#
# example access restrictions setup
#
com2sec readonly default public
group MyROGroup v1 readonly
view all included .1 80
access MyROGroup "" any noauth exact all none none
#
# enable master agent for AgentX subagents
#
master agentx
```

`/etc/frr/ospfd.conf`:

```
! ... the rest of ospfd.conf has been omitted for clarity ...
!
agentx
!
```

Upon successful connection, you should get something like this in the log of each FRR daemons:

```
2012/05/25 11:39:08 ZEBRA: snmp[info]: NET-SNMP version 5.4.3 AgentX subagent connected
```

Then, you can use the following command to check everything works as expected:

```
# snmpwalk -c public -v1 localhost .1.3.6.1.2.1.14.1.1
OSPF-MIB::ospfRouterId.0 = IpAddress: 192.168.42.109
[...]
```

An example below is how to query SNMP for BGP:

```
$ # BGP4-MIB (https://www.circitor.fr/Mibs/Mib/B/BGP4-MIB.mib)
$ snmpwalk -c public -v2c -On -Ln localhost .1.3.6.1.2.1.15

$ # BGP4V2-MIB (http://www.circitor.fr/Mibs/Mib/B/BGP4V2-MIB.mib)
$ # Information about the peers (bgp4V2PeerTable):
$ snmpwalk -c public -v2c -On -Ln localhost .1.3.6.1.3.5.1.1.2
...
.1.3.6.1.3.5.1.1.2.1.1.1.4.192.168.10.124 = Gauge32: 0
.1.3.6.1.3.5.1.1.2.1.1.2.16.42.2.71.128.10.188.0.0.0.0.0.0.0.0.0.2 = Gauge32: 0
.1.3.6.1.3.5.1.1.2.1.2.1.4.192.168.10.124 = INTEGER: 1
.1.3.6.1.3.5.1.1.2.1.2.2.16.42.2.71.128.10.188.0.0.0.0.0.0.0.0.0.2 = INTEGER: 2
.1.3.6.1.3.5.1.1.2.1.3.1.4.192.168.10.124 = Hex-STRING: C0 A8 0A 11
.1.3.6.1.3.5.1.1.2.1.3.2.16.42.2.71.128.10.188.0.0.0.0.0.0.0.0.0.2 = Hex-
↪STRING: 2A 02 47 80 0A BC 00 00 00 00 00 00 00 00 00 01
.1.3.6.1.3.5.1.1.2.1.4.1.4.192.168.10.124 = INTEGER: 1
```

(continues on next page)

(continued from previous page)

```
.1.3.6.1.3.5.1.1.2.1.4.2.16.42.2.71.128.10.188.0.0.0.0.0.0.0.0.0.2 = INTEGER: 2
.1.3.6.1.3.5.1.1.2.1.5.1.4.192.168.10.124 = Hex-STRING: C0 A8 0A 7C
.1.3.6.1.3.5.1.1.2.1.5.2.16.42.2.71.128.10.188.0.0.0.0.0.0.0.0.0.2 = Hex-
↪STRING: 2A 02 47 80 0A BC 00 00 00 00 00 00 00 00 00 02
.1.3.6.1.3.5.1.1.2.1.6.1.4.192.168.10.124 = Gauge32: 179
.1.3.6.1.3.5.1.1.2.1.6.2.16.42.2.71.128.10.188.0.0.0.0.0.0.0.0.0.2 = Gauge32:␣
↪179
.1.3.6.1.3.5.1.1.2.1.7.1.4.192.168.10.124 = Gauge32: 65002
.1.3.6.1.3.5.1.1.2.1.7.2.16.42.2.71.128.10.188.0.0.0.0.0.0.0.0.0.2 = Gauge32:␣
↪65002
.1.3.6.1.3.5.1.1.2.1.8.1.4.192.168.10.124 = Hex-STRING: C0 A8 0A 11
.1.3.6.1.3.5.1.1.2.1.8.2.16.42.2.71.128.10.188.0.0.0.0.0.0.0.0.0.2 = Hex-
↪STRING: C0 A8 0A 11
.1.3.6.1.3.5.1.1.2.1.9.1.4.192.168.10.124 = Gauge32: 41894
.1.3.6.1.3.5.1.1.2.1.9.2.16.42.2.71.128.10.188.0.0.0.0.0.0.0.0.0.2 = Gauge32:␣
↪39960
.1.3.6.1.3.5.1.1.2.1.10.1.4.192.168.10.124 = Gauge32: 65001
.1.3.6.1.3.5.1.1.2.1.10.2.16.42.2.71.128.10.188.0.0.0.0.0.0.0.0.0.2 = Gauge32:␣
↪65001
.1.3.6.1.3.5.1.1.2.1.11.1.4.192.168.10.124 = Hex-STRING: C8 C8 C8 CA
.1.3.6.1.3.5.1.1.2.1.11.2.16.42.2.71.128.10.188.0.0.0.0.0.0.0.0.0.2 = Hex-
↪STRING: C8 C8 C8 CA
.1.3.6.1.3.5.1.1.2.1.12.1.4.192.168.10.124 = INTEGER: 2
.1.3.6.1.3.5.1.1.2.1.12.2.16.42.2.71.128.10.188.0.0.0.0.0.0.0.0.0.2 = INTEGER:␣
↪2
.1.3.6.1.3.5.1.1.2.1.13.1.4.192.168.10.124 = INTEGER: 6
.1.3.6.1.3.5.1.1.2.1.13.2.16.42.2.71.128.10.188.0.0.0.0.0.0.0.0.0.2 = INTEGER:␣
↪6

$ # Information about the BGP table (bgp4V2NlriTable):
$ snmpwalk -c public -v2c -On -Ln localhost .1.3.6.1.3.5.1.1.9
...
.1.3.6.1.3.5.1.1.9.1.22.1.4.10.0.2.0.24.192.168.10.124 = Gauge32: 1
.1.3.6.1.3.5.1.1.9.1.22.1.4.10.10.100.0.24.192.168.10.124 = Gauge32: 1
.1.3.6.1.3.5.1.1.9.1.22.1.4.172.16.31.1.32.192.168.10.124 = Gauge32: 1
.1.3.6.1.3.5.1.1.9.1.22.1.4.172.16.31.2.32.192.168.10.124 = Gauge32: 1
.1.3.6.1.3.5.1.1.9.1.22.1.4.172.16.31.3.32.192.168.10.124 = Gauge32: 1
.1.3.6.1.3.5.1.1.9.1.22.1.4.192.168.0.0.24.192.168.10.124 = Gauge32: 1
.1.3.6.1.3.5.1.1.9.1.22.1.4.192.168.1.0.24.192.168.10.124 = Gauge32: 1
.1.3.6.1.3.5.1.1.9.1.22.1.4.192.168.10.0.24.192.168.10.124 = Gauge32: 1
.1.3.6.1.3.5.1.1.9.1.22.2.16.42.2.71.128.10.188.0.0.0.0.0.0.0.0.0.0.64.42.2.71.
↪128.10.188.0.0.0.0.0.0.0.0.0.2 = Gauge32: 1
.1.3.6.1.3.5.1.1.9.1.24.1.4.10.0.2.0.24.192.168.10.124 = Hex-STRING: 02 01 FD␣
↪E9
.1.3.6.1.3.5.1.1.9.1.24.1.4.10.10.100.0.24.192.168.10.124 = Hex-STRING: 02 01␣
↪FD E9
.1.3.6.1.3.5.1.1.9.1.24.1.4.172.16.31.1.32.192.168.10.124 = Hex-STRING: 02 01␣
↪FD E9
.1.3.6.1.3.5.1.1.9.1.24.1.4.172.16.31.2.32.192.168.10.124 = Hex-STRING: 02 01␣
↪FD E9
.1.3.6.1.3.5.1.1.9.1.24.1.4.172.16.31.3.32.192.168.10.124 = Hex-STRING: 02 01␣
↪FD E9
```

(continues on next page)

(continued from previous page)

```
.1.3.6.1.3.5.1.1.9.1.24.1.4.192.168.0.0.24.192.168.10.124 = Hex-STRING: 02 01␣
↪FD E9
.1.3.6.1.3.5.1.1.9.1.24.1.4.192.168.1.0.24.192.168.10.124 = Hex-STRING: 02 01␣
↪FD E9
.1.3.6.1.3.5.1.1.9.1.24.1.4.192.168.10.0.24.192.168.10.124 = Hex-STRING: 02 01␣
↪FD E9
.1.3.6.1.3.5.1.1.9.1.24.2.16.42.2.71.128.10.188.0.0.0.0.0.0.0.0.0.0.64.42.2.71.
↪128.10.188.0.0.0.0.0.0.0.0.0.2 = Hex-STRING: 02 01 FD E9
```

The AgentX protocol can be transported over a Unix socket or using TCP or UDP. It usually defaults to a Unix socket and depends on how NetSNMP was built. If need to configure FRR to use another transport, you can configure it through `/etc/snmp/frr.conf`:

```
[snmpd]
# Use a remote master agent
agentXSocket tcp:192.168.15.12:705
```

Here is the syntax for using AgentX:

agentx
Once enabled, it can't be unconfigured. Only removing from the daemons file the keyword `agentx` takes an effect.

2.10.4 Handling SNMP Traps

To handle snmp traps make sure your snmp setup of frr works correctly as described in the frr documentation in *SNMP Support*.

The BGP4 mib will send traps on peer up/down events. These should be visible in your snmp logs with a message similar to:

```
snmpd[13733]: Got trap from peer on fd 14
```

To react on these traps they should be handled by a trapsink. Configure your trapsink by adding the following lines to `/etc/snmpd/snmpd.conf`:

```
# send traps to the snmptrapd on localhost
trapsink localhost
```

This will send all traps to an snmptrapd running on localhost. You can of course also use a dedicated management station to catch traps. Configure the snmptrapd daemon by adding the following line to `/etc/snmpd/snmptrapd.conf`:

```
traphandle .1.3.6.1.4.1.3317.1.2.2 /etc/snmp/snmptrap_handle.sh
```

This will use the bash script `/etc/snmp/snmptrap_handle.sh` to handle the BGP4 traps. To add traps for other protocol daemons, lookup their appropriate OID from their mib. (For additional information about which traps are supported by your mib, lookup the mib on http://www.oidview.com/mibs/detail.html).

Make sure *snmptrapd* is started.

The snmptrap_handle.sh script I personally use for handling BGP4 traps is below. You can of course do all sorts of things when handling traps, like sound a siren, have your display flash, etc., be creative ;).

```
#!/bin/bash

# routers name
ROUTER=`hostname -s`

#email address use to sent out notification
EMAILADDR="john@doe.com"
#email address used (allongside above) where warnings should be sent
EMAILADDR_WARN="sms-john@doe.com"

# type of notification
TYPE="Notice"

# local snmp community for getting AS belonging to peer
COMMUNITY="<community>"

# if a peer address is in $WARN_PEERS a warning should be sent
WARN_PEERS="192.0.2.1"

# get stdin
INPUT=`cat -`

# get some vars from stdin
uptime=`echo $INPUT | cut -d' ' -f5`
peer=`echo $INPUT | cut -d' ' -f8 | sed -e 's/SNMPv2-SMI::mib-2.15.3.1.14.//g'`
peerstate=`echo $INPUT | cut -d' ' -f13`
errorcode=`echo $INPUT | cut -d' ' -f9 | sed -e 's/\\"//g'`
suberrorcode=`echo $INPUT | cut -d' ' -f10 | sed -e 's/\\"//g'`
remoteas=`snmpget -v2c -c $COMMUNITY localhost SNMPv2-SMI::mib-2.15.3.1.9.$peer | cut -d
↪' ' -f4`

WHOISINFO=`whois -h whois.ripe.net " -r AS$remoteas" | egrep '(as-name|descr)'`
asname=`echo "$WHOISINFO" | grep "^as-name:" | sed -e 's/^as-name://g' -e 's/  //g' -e
↪'s/^ //g' | uniq`
asdescr=`echo "$WHOISINFO" | grep "^descr:" | sed -e 's/^descr://g' -e 's/  //g' -e 's/^␣
↪//g' | uniq`

# if peer address is in $WARN_PEER, the email should also
# be sent to $EMAILADDR_WARN
for ip in $WARN_PEERS; do
if [ "x$ip" == "x$peer" ]; then
EMAILADDR="$EMAILADDR,$EMAILADDR_WARN"
TYPE="WARNING"
break
fi
done

# convert peer state
case "$peerstate" in
1) peerstate="Idle" ;;
2) peerstate="Connect" ;;
3) peerstate="Active" ;;
4) peerstate="Opensent" ;;
```

(continues on next page)

(continued from previous page)

```
5) peerstate="Openconfirm" ;;
6) peerstate="Established" ;;
*) peerstate="Unknown" ;;
esac

# get textual messages for errors
case "$errorcode" in
00)
error="No error"
suberror=""
;;
01)
error="Message Header Error"
case "$suberrorcode" in
01) suberror="Connection Not Synchronized" ;;
02) suberror="Bad Message Length" ;;
03) suberror="Bad Message Type" ;;
*) suberror="Unknown" ;;
esac
;;
02)
error="OPEN Message Error"
case "$suberrorcode" in
01) suberror="Unsupported Version Number" ;;
02) suberror="Bad Peer AS" ;;
03) suberror="Bad BGP Identifier" ;;
04) suberror="Unsupported Optional Parameter" ;;
05) suberror="Authentication Failure" ;;
06) suberror="Unacceptable Hold Time" ;;
*) suberror="Unknown" ;;
esac
;;
03)
error="UPDATE Message Error"
case "$suberrorcode" in
01) suberror="Malformed Attribute List" ;;
02) suberror="Unrecognized Well-known Attribute" ;;
03) suberror="Missing Well-known Attribute" ;;
04) suberror="Attribute Flags Error" ;;
05) suberror="Attribute Length Error" ;;
06) suberror="Invalid ORIGIN Attribute" ;;
07) suberror="AS Routing Loop" ;;
08) suberror="Invalid NEXT_HOP Attribute" ;;
09) suberror="Optional Attribute Error" ;;
10) suberror="Invalid Network Field" ;;
11) suberror="Malformed AS_PATH" ;;
*) suberror="Unknown" ;;
esac
;;
04)
error="Hold Timer Expired"
suberror=""
```

(continues on next page)

(continued from previous page)

```
;;
05)
error="Finite State Machine Error"
suberror=""
;;
06)
error="Cease"
case "$suberrorcode" in
01) suberror="Maximum Number of Prefixes Reached" ;;
02) suberror="Administrative Shutdown" ;;
03) suberror="Peer De-configured" ;;
04) suberror="Administrative Reset" ;;
05) suberror="Connection Rejected" ;;
06) suberror="Other Configuration Change" ;;
07) suberror="Connection Collision Resolution" ;;
08) suberror="Out of Resources" ;;
09) suberror="MAX" ;;
*) suberror="Unknown" ;;
esac
;;
*)
error="Unknown"
suberror=""
;;
esac

# create textual message from errorcodes
if [ "x$suberror" == "x" ]; then
NOTIFY="$errorcode ($error)"
else
NOTIFY="$errorcode/$suberrorcode ($error/$suberror)"
fi

# form a decent subject
SUBJECT="$TYPE: $ROUTER [bgp] $peer is $peerstate: $NOTIFY"
# create the email body
MAIL=`cat << EOF
BGP notification on router $ROUTER.

Peer: $peer
AS: $remoteas
New state: $peerstate
Notification: $NOTIFY

Info:
$asname
$asdescr

Snmpd uptime: $uptime
EOF`

# mail the notification
```

(continues on next page)

(continued from previous page)

```
echo "$MAIL" | mail -s "$SUBJECT" $EMAILADDR
```

2.11 Scripting

The behavior of FRR may be extended or customized using its built-in scripting capabilities. The scripting language is Lua 5.3. This guide assumes Lua knowledge. For more information on Lua, consult the Lua 5.3 reference manual, or *Programming in Lua* (note that the free version covers only Lua 5.0).

https://www.lua.org/manual/5.3/

http://www.lua.org/pil/contents.html

2.11.1 Scripting

See also:

Developer docs for scripting

How to use

1. Identify the Lua function name. See *Available Lua hook calls*.
2. Write the Lua script
3. Configure FRR to use the Lua script

In order to use scripting, FRR must be built with `--enable-scripting`.

Note: Scripts are typically loaded just-in-time. This means you can change the contents of a script that is in use without restarting FRR. Not all scripting locations may behave this way; refer to the documentation for the particular location.

Example: on_rib_process_dplane_results

This example shows how to write a Lua script that logs changes when a route is added.

First, identify the Lua hook call to attach a Lua function to: this will be the name of the Lua function. In this case, since the hook call is *on_rib_process_dplane_results*:

```
function on_rib_process_dplane_results(ctx)
   log.info(ctx.rinfo.zd_dest.network)
   return {}
```

The documentation for *on_rib_process_dplane_results* tells us its arguments. Here, the destination prefix for a route is being logged out.

Scripts live in `/etc/frr/scripts/` by default. This is configurable at compile time via `--with-scriptdir`. It may be overridden at runtime with the `--scriptdir` daemon option.

The documentation for *on_rib_process_dplane_results* indicates that the `script` command should be used to set the script. Assuming that the above function was created in `/etc/frr/scripts/my_dplane_script.lua`, the following vtysh command sets the script for the hook call:

```
script on_rib_process_dplane_results my_dplane_script
```

After the script is set, when the hook call is hit, FRR will look for a *on_rib_process_dplane_results* function in `/etc/frr/scripts/my_dplane_script.lua` and run it with the `ctx` object as its argument.

2.11.2 Available Lua hook calls

on_rib_process_dplane_results

2.12 Nexthop Groups

Nexthop groups are a way to encapsulate ECMP information together. It's a listing of ECMP nexthops used to forward packets.

`nexthop-group NAME`

Create a nexthop-group with an associated NAME. This will put you into a sub-mode where you can specify individual nexthops. To exit this mode type exit or end as per normal conventions for leaving a sub-mode.

`nexthop [A.B.C.D|X:X::X:XX] [interface [onlink]] [nexthop-vrf NAME] [label LABELS]`

Create a v4 or v6 nexthop. All normal rules for creating nexthops that you are used to are allowed here. The syntax was intentionally kept the same as creating nexthops as you would for static routes.

`resilient buckets (1-256) idle-timer (1-4294967295) unbalanced-timer (1-4294967295)`

Create a resilient Nexthop Group with the specified number of buckets, and associated timers. Instead of using the normal kernel hashing methodology this specifies that X buckets will be created for the nexthop group and when a nexthop is lost the buckets forwarding that particular nexthop will be automatically re-assigned. This cli command must be the first command entered currently. Additionally this command only works with linux 5.19 kernels or newer.

CHAPTER THREE

PROTOCOLS

3.1 Zebra

zebra is an IP routing manager. It provides kernel routing table updates, interface lookups, and redistribution of routes between different routing protocols.

3.1.1 Invoking zebra

Besides the common invocation options (*Common Invocation Options*), the *zebra* specific invocation options are listed below.

`-b, --batch`
: Runs in batch mode. *zebra* parses configuration file and terminates immediately.

`-K TIME, --graceful_restart TIME`
: If this option is specified, the graceful restart time is TIME seconds. Zebra, when started, will read in routes. Those routes that Zebra identifies that it was the originator of will be swept in TIME seconds. If no time is specified then we will sweep those routes immediately. Under the *BSD's, there is no way to properly store the originating route and the route types in this case will show up as a static route with an admin distance of 255.

`-r, --retain`
: When program terminates, do not flush routes installed by *zebra* from the kernel.

`-e X, --ecmp X`
: Run zebra with a limited ecmp ability compared to what it is compiled to. If you are running zebra on hardware limited functionality you can force zebra to limit the maximum ecmp allowed to X. This number is bounded by what you compiled FRR with as the maximum number.

`-n, --vrfwnetns`
: When *Zebra* starts with this option, the VRF backend is based on Linux network namespaces. That implies that all network namespaces discovered by ZEBRA will create an associated VRF. The other daemons will operate on the VRF VRF defined by *Zebra*, as usual.

See also:

Virtual Routing and Forwarding

`-z <path_to_socket>, --socket <path_to_socket>`
: If this option is supplied on the cli, the path to the zebra control socket(zapi), is used. This option overrides a -N <namespace> option if handed to it on the cli.

`--v6-rr-semantics`
: The linux kernel is receiving the ability to use the same route replacement semantics for v6 that v4 uses. If you are using a kernel that supports this functionality then run *Zebra* with this option and we will use Route Replace Semantics instead of delete than add.

--asic-offload=[notify_on_offload|notify_on_ack]
: The linux kernel has the ability to use asic-offload (see switchdev development). When the operator knows that FRR will be working in this way, allow them to specify this with FRR. At this point this code only supports asynchronous notification of the offload state. In other words the initial ACK received for linux kernel installation does not give zebra any data about what the state of the offload is. This option takes the optional parameters notify_on_offload or notify_on_ack. This signals to zebra to notify upper level protocols about route installation/update on ack received from the linux kernel or from offload notification.

-s <SIZE>, **--nl-bufsize** <SIZE>
: Allow zebra to modify the default receive buffer size to SIZE in bytes. Under *BSD only the -s option is available.

3.1.2 Configuration Addresses behaviour

At startup, *Zebra* will first discover the underlying networking objects from the operating system. This includes interfaces, addresses of interfaces, static routes, etc. Then, it will read the configuration file, including its own interface addresses, static routes, etc. All this information comprises the operational context from *Zebra*. But configuration context from *Zebra* will remain the same as the one from `zebra.conf` config file. As an example, executing the following `show running-config` will reflect what was in `zebra.conf`. In a similar way, networking objects that are configured outside of the *Zebra* like *iproute2* will not impact the configuration context from *Zebra*. This behaviour permits you to continue saving your own config file, and decide what is really to be pushed on the config file, and what is dependent on the underlying system. Note that inversely, from *Zebra*, you will not be able to delete networking objects that were previously configured outside of *Zebra*.

3.1.3 Interface Commands

Standard Commands

interface IFNAME

interface IFNAME vrf VRF

shutdown
: Up or down the current interface.

ip address ADDRESS/PREFIX

ipv6 address ADDRESS/PREFIX
: Set the IPv4 or IPv6 address/prefix for the interface.

ip address LOCAL-ADDR peer PEER-ADDR/PREFIX
: Configure an IPv4 Point-to-Point address on the interface. (The concept of PtP addressing does not exist for IPv6.)

  `local-addr` has no subnet mask since the local side in PtP addressing is always a single (/32) address. `peer-addr/prefix` can be an arbitrary subnet behind the other end of the link (or even on the link in Point-to-Multipoint setups), though generally /32s are used.

description DESCRIPTION ...
: Set description for the interface.

mpls enable
: Enable or disable mpls kernel processing on the interface, for linux. Interfaces configured with mpls will not automatically turn on if mpls kernel modules do not happen to be loaded. This command will fail on 3.X linux kernels and does not work on non-linux systems at all.

multicast
: Enable or disable multicast flag for the interface.

`bandwidth (1-10000000)`
Set bandwidth value of the interface in kilobits/sec. This is for calculating OSPF cost. This command does not affect the actual device configuration.

`link-detect`
Enable or disable link-detect on platforms which support this. Currently only Linux, and only where network interface drivers support reporting link-state via the `IFF_RUNNING` flag.

In FRR, link-detect is on by default.

Link Parameters Commands

Note: At this time, FRR offers partial support for some of the routing protocol extensions that can be used with MPLS-TE. FRR does not support a complete RSVP-TE solution currently.

`link-params`
Enter into the link parameters sub node. At least 'enable' must be set to activate the link parameters, and consequently routing information that could be used as part of Traffic Engineering on this interface. MPLS-TE must be enable at the OSPF (*Traffic Engineering*) or ISIS (*Traffic Engineering*) router level in complement to this.

Under link parameter statement, the following commands set the different TE values:

`enable`
Enable link parameters for this interface.

`metric (0-4294967295)`

`max-bw BANDWIDTH`

`max-rsv-bw BANDWIDTH`

`unrsv-bw (0-7) BANDWIDTH`
These commands specifies the Traffic Engineering parameters of the interface in conformity to RFC3630 (OSPF) or RFC5305 (ISIS). There are respectively the TE Metric (different from the OSPF or ISIS metric), Maximum Bandwidth (interface speed by default), Maximum Reservable Bandwidth, Unreserved Bandwidth for each 0-7 priority and Admin Group (ISIS) or Resource Class/Color (OSPF).

Note that BANDWIDTH is specified in IEEE floating point format and express in Bytes/second.

`admin-grp 0x(0-FFFFFFFF)`
This commands configures the Traffic Engineering Admin-Group of the interface as specified in RFC3630 (OSPF) or RFC5305 (ISIS). Admin-group is also known as Resource Class/Color in the OSPF protocol.

`[no] affinity AFFINITY-MAP-NAME`
This commands configures the Traffic Engineering Admin-Group of the interface using the affinity-map definitions (*Affinity Maps*). Multiple AFFINITY-MAP-NAME can be specified at the same time. Affinity-map names are added or removed if `no` is present. It means that specifying one value does not override the full list.

`admin-grp` and `affinity` commands provide two ways of setting admin-groups. They cannot be both set on the same interface.

`[no] affinity-mode [extended|standard|both]`
This commands configures which admin-group format is set by the affinity command. `extended` Admin-Group is the default and uses the RFC7308 format. `standard` mode uses the standard admin-group format that is defined by RFC3630, RFC5305 and RFC5329. When the `standard` mode is set, affinity-maps with bit-positions higher than 31 cannot be applied to the interface. The `both` mode allows setting standard and extended admin-group on the link at the same time. In this case, the bit-positions 0 to 31 are the same on standard and extended admin-groups.

Note that extended admin-groups are only supported by IS-IS for the moment.

`delay (0-16777215) [min (0-16777215) | max (0-16777215)]`

`delay-variation (0-16777215)`

`packet-loss PERCENTAGE`

`res-bw BANDWIDTH`

`ava-bw BANDWIDTH`

`use-bw BANDWIDTH`

These command specifies additional Traffic Engineering parameters of the interface in conformity to draft-ietf-ospf-te-metrics-extension-05.txt and draft-ietf-isis-te-metrics-extension-03.txt. There are respectively the delay, jitter, loss, available bandwidth, reservable bandwidth and utilized bandwidth.

Note that BANDWIDTH is specified in IEEE floating point format and express in Bytes/second. Delays and delay variation are express in micro-second (µs). Loss is specified in PERCENTAGE ranging from 0 to 50.331642% by step of 0.000003.

`neighbor <A.B.C.D> as (0-65535)`

Specifies the remote ASBR IP address and Autonomous System (AS) number for InterASv2 link in OSPF (RFC5392). Note that this option is not yet supported for ISIS (RFC5316).

Global Commands

`zebra protodown reason-bit (0-31)`

This command is only supported for linux and a kernel > 5.1. Change reason-bit frr uses for setting protodown. We default to 7, but if another userspace app ever conflicts with this, you can change it here. The descriptor for this bit should exist in `/etc/iproute2/protodown_reasons.d/` to display with `ip -d link show`.

3.1.4 Nexthop Tracking

Nexthop tracking doesn't resolve nexthops via the default route by default. Allowing this might be useful when e.g. you want to allow BGP to peer across the default route.

`zebra nexthop-group keep (1-3600)`

Set the time that zebra will keep a created and installed nexthop group before removing it from the system if the nexthop group is no longer being used. The default time is 180 seconds.

`ip nht resolve-via-default`

Allow IPv4 nexthop tracking to resolve via the default route. This parameter is configured per-VRF, so the command is also available in the VRF subnode.

`ipv6 nht resolve-via-default`

Allow IPv6 nexthop tracking to resolve via the default route. This parameter is configured per-VRF, so the command is also available in the VRF subnode.

`show ip nht [vrf NAME] [A.B.C.D|X:X::X:X] [mrib] [json]`

Show nexthop tracking status for address resolution. If vrf is not specified then display the default vrf. If `all` is specified show all vrf address resolution output. If an ipv4 or ipv6 address is not specified then display all addresses tracked, else display the requested address. The mrib keyword indicates that the operator wants to see the multicast rib address resolution table. An alternative form of the command is `show ip import-check` and this form of the command is deprecated at this point in time. User can get that information as JSON string when `json` key word at the end of cli is presented.

`show ip nht route-map [vrf <NAME|all>] [json]`

This command displays route-map attach point to nexthop tracking and displays list of protocol with its applied route-map. When zebra considers sending NHT resoultion, the nofification only sent to appropriate client protocol only after applying route-map filter. User can get that information as JSON format when `json` keyword at the end of cli is presented.

3.1.5 PBR dataplane programming

Some dataplanes require the PBR nexthop to be resolved into a SMAC, DMAC and outgoing interface

`pbr nexthop-resolve`

Resolve PBR nexthop via ip neigh tracking

3.1.6 Administrative Distance

Administrative distance allows FRR to make decisions about what routes should be installed in the rib based upon the originating protocol. The lowest Admin Distance is the route selected. This is purely a subjective decision about ordering and care has been taken to choose the same distances that other routing suites have chosen.

Protocol	Distance
System	0
Kernel	0
Connect	0
Static	1
NHRP	10
EBGP	20
EIGRP	90
BABEL	100
OSPF	110
ISIS	115
OPENFABRIC	115
RIP	120
Table	150
SHARP	150
IBGP	200
PBR	200

An admin distance of 255 indicates to Zebra that the route should not be installed into the Data Plane. Additionally routes with an admin distance of 255 will not be redistributed.

Zebra does treat Kernel routes as special case for the purposes of Admin Distance. Upon learning about a route that is not originated by FRR we read the metric value as a uint32_t. The top byte of the value is interpreted as the Administrative Distance and the low three bytes are read in as the metric. This special case is to facilitate VRF default routes.

3.1.7 Route Replace Semantics

When using the Linux Kernel as a forwarding plane, routes are installed with a metric of 20 to the kernel. Please note that the kernel's metric value bears no resemblence to FRR's RIB metric or admin distance. It merely is a way for the Linux Kernel to decide which route to use if it has multiple routes for the same prefix from multiple sources. An example here would be if someone else was running another routing suite besides FRR at the same time, the kernel must choose what route to use to forward on. FRR choose the value of 20 because of two reasons. FRR wanted a value small enough to be chosen but large enough that the operator could allow route prioritization by the kernel when multiple routing suites are being run and FRR wanted to take advantage of Route Replace semantics that the linux kernel offers. In order for Route Replacement semantics to work FRR must use the same metric when issuing the replace command. Currently FRR only supports Route Replace semantics using the Linux Kernel.

3.1.8 Virtual Routing and Forwarding

FRR supports VRF (Virtual Routing and Forwarding). VRF is a way to separate networking contexts on the same machine. Those networking contexts are associated with separate interfaces, thus making it possible to associate one interface with a specific VRF.

VRF can be used, for example, when instantiating per enterprise networking services, without having to instantiate the physical host machine or the routing management daemons for each enterprise. As a result, interfaces are separate for each set of VRF, and routing daemons can have their own context for each VRF.

This conceptual view introduces the *Default VRF* case. If the user does not configure any specific VRF, then by default, FRR uses the *Default VRF*. The name "default" is used to refer to this VRF in various CLI commands and YANG models. It is possible to change that name by passing the `-o` option to all daemons, for example, one can use `-o vrf0` to change the name to "vrf0". The easiest way to pass the same option to all daemons is to use the `frr_global_options` variable in the *Daemons Configuration File*.

Configuring VRF networking contexts can be done in various ways on FRR. The VRF interfaces can be configured by entering in interface configuration mode `interface IFNAME vrf VRF`.

A VRF backend mode is chosen when running *Zebra*.

If no option is chosen, then the *Linux VRF* implementation as references in https://www.kernel.org/doc/Documentation/networking/vrf.txt will be mapped over the *Zebra* VRF. The routing table associated to that VRF is a Linux table identifier located in the same *Linux network namespace* where *Zebra* started. Please note when using the *Linux VRF* routing table it is expected that a default Kernel route will be installed that has a metric as outlined in the www.kernel.org doc above. The Linux Kernel does table lookup via a combination of rule application of the rule table and then route lookup of the specified table. If no route match is found then the next applicable rule is applied to find the next route table to use to look for a route match. As such if your VRF table does not have a default blackhole route with a high metric VRF route lookup will leave the table specified by the VRF, which is undesirable.

If the `-n` option is chosen, then the *Linux network namespace* will be mapped over the *Zebra* VRF. That implies that *Zebra* is able to configure several *Linux network namespaces*. The routing table associated to that VRF is the whole routing tables located in that namespace. For instance, this mode matches OpenStack Network Namespaces. It matches also OpenFastPath. The default behavior remains Linux VRF which is supported by the Linux kernel community, see https://www.kernel.org/doc/Documentation/networking/vrf.txt.

Because of that difference, there are some subtle differences when running some commands in relationship to VRF. Here is an extract of some of those commands:

`vrf VRF`

This command is available on configuration mode. By default, above command permits accessing the VRF configuration mode. This mode is available for both VRFs. It is to be noted that *Zebra* does not create Linux VRF. The network administrator can however decide to provision this command in configuration file to provide more clarity about the intended configuration.

netns NAMESPACE

This command is based on VRF configuration mode. This command is available when *Zebra* is run in `-n` mode. This command reflects which *Linux network namespace* is to be mapped with *Zebra* VRF. It is to be noted that *Zebra* creates and detects added/suppressed VRFs from the Linux environment (in fact, those managed with iproute2). The network administrator can however decide to provision this command in configuration file to provide more clarity about the intended configuration.

show ip route vrf VRF

The show command permits dumping the routing table associated to the VRF. If *Zebra* is launched with default settings, this will be the `TABLENO` of the VRF configured on the kernel, thanks to information provided in https://www.kernel.org/doc/Documentation/networking/vrf.txt. If *Zebra* is launched with `-n` option, this will be the default routing table of the *Linux network namespace* `VRF`.

show ip route vrf VRF table TABLENO

The show command is only available with `-n` option. This command will dump the routing table `TABLENO` of the *Linux network namespace* `VRF`.

show ip route vrf VRF tables

This command will dump the routing tables within the vrf scope. If `vrf all` is executed, all routing tables will be dumped.

show <ip|ipv6> route summary [vrf VRF] [table TABLENO] [prefix]

This command will dump a summary output of the specified VRF and TABLENO combination. If neither VRF or TABLENO is specified FRR defaults to the default vrf and default table. If prefix is specified dump the number of prefix routes.

3.1.9 Table Allocation

Some services like BGP flowspec allocate routing tables to perform policy routing based on netfilter criteria and IP rules. In order to avoid conflicts between VRF allocated routing tables and those services, Zebra proposes to define a chunk of routing tables to use by other services.

Allocation configuration can be done like below, with the range of the chunk of routing tables to be used by the given service.

ip table range <STARTTABLENO> <ENDTABLENO>

3.1.10 ECMP

FRR supports ECMP as part of normal operations and is generally compiled with a limit of 64 way ECMP. This of course can be modified via configure options on compilation if the end operator desires to do so. Individual protocols each have their own way of dictating ECMP policy and their respective documentation should be read.

ECMP can be inspected in zebra by doing a `show ip route X` command.

```
eva# show ip route 4.4.4.4/32
Codes: K - kernel route, C - connected, S - static, R - RIP,
       O - OSPF, I - IS-IS, B - BGP, E - EIGRP, N - NHRP,
       T - Table, v - VNC, V - VNC-Direct, A - Babel, D - SHARP,
       F - PBR, f - OpenFabric,
       > - selected route, * - FIB route, q - queued, r - rejected, b - backup
       t - trapped, o - offload failure

D>* 4.4.4.4/32 [150/0] via 192.168.161.1, enp39s0, weight 1, 00:00:02
  *                    via 192.168.161.2, enp39s0, weight 1, 00:00:02
```

(continues on next page)

(continued from previous page)

```
  *                     via 192.168.161.3, enp39s0, weight 1, 00:00:02
  *                     via 192.168.161.4, enp39s0, weight 1, 00:00:02
  *                     via 192.168.161.5, enp39s0, weight 1, 00:00:02
  *                     via 192.168.161.6, enp39s0, weight 1, 00:00:02
  *                     via 192.168.161.7, enp39s0, weight 1, 00:00:02
  *                     via 192.168.161.8, enp39s0, weight 1, 00:00:02
  *                     via 192.168.161.9, enp39s0, weight 1, 00:00:02
  *                     via 192.168.161.10, enp39s0, weight 1, 00:00:02
  *                     via 192.168.161.11, enp39s0, weight 1, 00:00:02
  *                     via 192.168.161.12, enp39s0, weight 1, 00:00:02
  *                     via 192.168.161.13, enp39s0, weight 1, 00:00:02
  *                     via 192.168.161.14, enp39s0, weight 1, 00:00:02
  *                     via 192.168.161.15, enp39s0, weight 1, 00:00:02
  *                     via 192.168.161.16, enp39s0, weight 1, 00:00:02
```

In this example we have 16 way ecmp for the 4.4.4.4/32 route. The * character tells us that the route is installed in the Data Plane, or FIB.

If you are using the Linux kernel as a Data Plane, this can be inspected via a `ip route show X` command:

```
sharpd@eva ~/f/doc(ecmp_doc_change)> ip route show 4.4.4.4/32
4.4.4.4 nhid 185483868 proto sharp metric 20
   nexthop via 192.168.161.1 dev enp39s0 weight 1
   nexthop via 192.168.161.10 dev enp39s0 weight 1
   nexthop via 192.168.161.11 dev enp39s0 weight 1
   nexthop via 192.168.161.12 dev enp39s0 weight 1
   nexthop via 192.168.161.13 dev enp39s0 weight 1
   nexthop via 192.168.161.14 dev enp39s0 weight 1
   nexthop via 192.168.161.15 dev enp39s0 weight 1
   nexthop via 192.168.161.16 dev enp39s0 weight 1
   nexthop via 192.168.161.2 dev enp39s0 weight 1
   nexthop via 192.168.161.3 dev enp39s0 weight 1
   nexthop via 192.168.161.4 dev enp39s0 weight 1
   nexthop via 192.168.161.5 dev enp39s0 weight 1
   nexthop via 192.168.161.6 dev enp39s0 weight 1
   nexthop via 192.168.161.7 dev enp39s0 weight 1
   nexthop via 192.168.161.8 dev enp39s0 weight 1
   nexthop via 192.168.161.9 dev enp39s0 weight 1
```

Once installed into the FIB, FRR currently has little control over what nexthops are chosen to forward packets on. Currently the Linux kernel has a `fib_multipath_hash_policy` sysctl which dictates how the hashing algorithm is used to forward packets.

3.1.11 Single Vxlan Device Support

FRR supports configuring VLAN-to-VNI mappings for EVPN-VXLAN, when working with the Linux kernel. In this new way, the mapping of a VLAN to a VNI is configured against a container VXLAN interface which is referred to as a ‘Single VXLAN device (SVD)’. Multiple VLAN to VNI mappings can be configured against the same SVD. This allows for a significant scaling of the number of VNIs since a separate VXLAN interface is no longer required for each VNI. Sample configuration of SVD with VLAN to VNI mappings is shown below.

If you are using the Linux kernel as a Data Plane, this can be configured via *ip link*, *bridge link* and *bridge vlan* commands:

```
# linux shell
ip link add dev bridge type bridge
ip link set dev bridge type bridge vlan_filtering 1
ip link add dev vxlan0 type vxlan external
ip link set dev vxlan0 master bridge
bridge link set dev vxlan0 vlan_tunnel on
bridge vlan add dev vxlan0 vid 100
bridge vlan add dev vxlan0 vid 100 tunnel_info id 100
bridge vlan tunnelshow
 port    vlan ids        tunnel id
 bridge  None
 vxlan0   100     100
```

show evpn access-vlan [IFNAME VLAN-ID | detail] [json]

Show information for EVPN Access VLANs.

```
VLAN            SVI              L2-VNI   VXLAN-IF        # Members
bridge.20       vlan20           20       vxlan0          0
bridge.10       vlan10           0        vxlan0          0
```

3.1.12 MPLS Commands

You can configure static mpls entries in zebra. Basically, handling MPLS consists of popping, swapping or pushing labels to IP packets.

MPLS Acronyms

LSR (Labeled Switch Router) Networking devices handling labels used to forward traffic between and through them.

LER (Labeled Edge Router) A Labeled edge router is located at the edge of an MPLS network, generally between an IP network and an MPLS network.

MPLS Push Action

The push action is generally used for LER devices, which want to encapsulate all traffic for a wished destination into an MPLS label. This action is stored in routing entry, and can be configured like a route:

`ip route NETWORK MASK GATEWAY|INTERFACE label LABEL`

NETWORK and MASK stand for the IP prefix entry to be added as static route entry. GATEWAY is the gateway IP address to reach, in order to reach the prefix. INTERFACE is the interface behind which the prefix is located. LABEL is the MPLS label to use to reach the prefix abovementioned.

You can check that the static entry is stored in the zebra RIB database, by looking at the presence of the entry.

```
zebra(configure)# ip route 1.1.1.1/32 10.0.1.1 label 777
zebra# show ip route
Codes: K - kernel route, C - connected, S - static, R - RIP,
O - OSPF, I - IS-IS, B - BGP, E - EIGRP, N - NHRP,
T - Table, v - VNC, V - VNC-Direct, A - Babel, D - SHARP,
F - PBR,
> - selected route, * - FIB route

S>* 1.1.1.1/32 [1/0] via 10.0.1.1, r2-eth0, label 777, 00:39:42
```

MPLS Swap and Pop Action

The swap action is generally used for LSR devices, which swap a packet with a label, with an other label. The Pop action is used on LER devices, at the termination of the MPLS traffic; this is used to remove MPLS header.

`mpls lsp INCOMING_LABEL GATEWAY OUTGOING_LABEL|explicit-null|implicit-null`

INCOMING_LABEL and OUTGOING_LABEL are MPLS labels with values ranging from 16 to 1048575. GATEWAY is the gateway IP address where to send MPLS packet. The outgoing label can either be a value or have an explicit-null label header. This specific header can be read by IP devices. The incoming label can also be removed; in that case the implicit-null keyword is used, and the outgoing packet emitted is an IP packet without MPLS header.

You can check that the MPLS actions are stored in the zebra MPLS table, by looking at the presence of the entry.

`show mpls table`

```
zebra(configure)# mpls lsp 18 10.125.0.2 implicit-null
zebra(configure)# mpls lsp 19 10.125.0.2 20
zebra(configure)# mpls lsp 21 10.125.0.2 explicit-null
zebra# show mpls table
Inbound                            Outbound
Label     Type          Nexthop     Label
--------  -------  ---------------  --------
18       Static       10.125.0.2  implicit-null
19       Static       10.125.0.2  20
21       Static       10.125.0.2  IPv4 Explicit Null
```

3.1.13 Segment-Routing IPv6

Segment-Routing is source routing paradigm that allows network operator to encode network intent into the packets. SRv6 is an implementation of Segment-Routing with application of IPv6 and segment-routing-header.

All routing daemon can use the Segment-Routing base framework implemented on zebra to use SRv6 routing mechanism. In that case, user must configure initial srv6 setting on FRR's cli or frr.conf or zebra.conf. This section shows how to configure SRv6 on FRR. Of course SRv6 can be used as standalone, and this section also helps that case.

show segment-routing srv6 locator [json]

This command dump SRv6-locator configured on zebra. SRv6-locator is used to route to the node before performing the SRv6-function. and that works as aggregation of SRv6-function's IDs. Following console log shows two SRv6-locators loc1 and loc2. All locators are identified by unique IPv6 prefix. User can get that information as JSON string when `json` key word at the end of cli is presented.

```
router# sh segment-routing srv6 locator
Locator:
Name                 ID      Prefix                   Status
-------------------- ------- ------------------------ -------
loc1                       1 2001:db8:1:1::/64        Up
loc2                       2 2001:db8:2:2::/64        Up
```

show segment-routing srv6 locator NAME detail [json]

As shown in the example, by specifying the name of the locator, you can see the detailed information for each locator. Locator can be represented by a single IPv6 prefix, but SRv6 is designed to share this Locator among multiple Routing Protocols. For this purpose, zebra divides the IPv6 prefix block that makes the Locator unique into multiple chunks, and manages the ownership of each chunk.

For example, loc1 has system as its owner. For example, loc1 is owned by system, which means that it is not yet proprietary to any routing protocol. For example, loc2 has sharp as its owner. This means that the shaprd for function development holds the owner of the chunk of this locator, and no other routing protocol will use this area.

```
router# show segment-routing srv6 locator loc1 detail
Name: loc1
Prefix: 2001:db8:1:1::/64
Chunks:
- prefix: 2001:db8:1:1::/64, owner: system

router# show segment-routing srv6 locator loc2 detail
Name: loc2
Prefix: 2001:db8:2:2::/64
Chunks:
- prefix: 2001:db8:2:2::/64, owner: sharp
```

segment-routing

Move from configure mode to segment-routing node.

srv6

Move from segment-routing node to srv6 node.

locators

Move from srv6 node to locator node. In this locator node, user can configure detailed settings such as the actual srv6 locator.

locator NAME

Create a new locator. If the name of an existing locator is specified, move to specified locator's configuration node to change the settings it.

prefix X:X::X:X/M [func-bits (0-64)] [block-len 40] [node-len 24]

Set the ipv6 prefix block of the locator. SRv6 locator is defined by RFC8986. The actual routing protocol specifies the locator and allocates a SID to be used by each routing protocol. This SID is included in the locator as an IPv6 prefix.

Following example console log shows the typical configuration of SRv6 data-plane. After a new SRv6 locator, named loc1, is created, loc1's prefix is configured as `2001:db8:1:1::/64`. If user or some routing daemon allocates new SID on this locator, new SID will allocated in range of this prefix. For example, if some routing daemon creates new SID on locator (`2001:db8:1:1::/64`), Then new SID will be `2001:db8:1:1:7::/80`, `2001:db8:1:1:8::/80`, and so on. Each locator has default SID that is SRv6 local function "End". Usually default SID is allocated as `PREFIX:1::`. (`PREFIX` is locator's prefix) For example, if user configure the locator's prefix as `2001:db8:1:1::/64`, then default SID will be `2001:db8:1:1:1::`)

This command takes three optional parameters: `func-bits`, `block-len` and `node-len`. These parameters allow users to set the format for the SIDs allocated from the SRv6 Locator. SID Format is defined in RFC 8986.

According to RFC 8986, an SRv6 SID consists of BLOCK:NODE:FUNCTION:ARGUMENT, where BLOCK is the SRv6 SID block (i.e., the IPv6 prefix allocated for SRv6 SIDs by the operator), NODE is the identifier of the parent node instantiating the SID, FUNCTION identifies the local behavior associated to the SID and ARGUMENT encodes additional information used to process the behavior. BLOCK and NODE make up the SRv6 Locator.

The function bits range is 16bits by default. If operator want to change function bits range, they can configure with `func-bits` option.

The `block-len` and `node-len` parameters allow the user to configure the length of the SRv6 SID block and SRv6 SID node, respectively. Both the lengths are expressed in bits.

`block-len`, `node-len` and `func-bits` may be any value as long as `block-len+node-len = locator-len` and `block-len+node-len+func-bits <= 128`.

When both `block-len` and `node-len` are omitted, the following default values are used: `block-len = 24`, `node-len = prefix-len-24`.

If only one parameter is omitted, the other parameter is derived from the first.

```
router# configure terminal
router(config)# segment-routinig
router(config-sr)# srv6
router(config-srv6)# locators
router(config-srv6-locs)# locator loc1
router(config-srv6-loc)# prefix 2001:db8:1:1::/64

router(config-srv6-loc)# show run
...
segment-routing
 srv6
  locators
   locator loc1
    prefix 2001:db8:1:1::/64
   !
...
```

behavior usid

Specify the SRv6 locator as a Micro-segment (uSID) locator. When a locator is specified as a uSID locator, all the SRv6 SIDs allocated from the locator by the routing protocols are bound to the SRv6 uSID behaviors. For example, if you configure BGP to use a locator specified as a uSID locator, BGP instantiates and advertises

SRv6 uSID behaviors (e.g., `uDT4` / `uDT6` / `uDT46`) instead of classic SRv6 behaviors (e.g., `End.DT4` / `End.DT6` / `End.DT46`).

```
router# configure terminal
router(config)# segment-routinig
router(config-sr)# srv6
router(config-srv6)# locators
router(config-srv6-locators)# locator loc1
router(config-srv6-locator)# prefix fc00:0:1::/48 block-len 32 node-len 16 func-bits 16
router(config-srv6-locator)# behavior usid

router(config-srv6-locator)# show run
...
segment-routing
 srv6
  locators
   locator loc1
    prefix fc00:0:1::/48
    behavior usid
   !
...
```

3.1.14 Multicast RIB Commands

The Multicast RIB provides a separate table of unicast destinations which is used for Multicast Reverse Path Forwarding decisions. It is used with a multicast source's IP address, hence contains not multicast group addresses but unicast addresses.

This table is fully separate from the default unicast table. However, RPF lookup can include the unicast table.

WARNING: RPF lookup results are non-responsive in this version of FRR, i.e. multicast routing does not actively react to changes in underlying unicast topology!

`ip multicast rpf-lookup-mode MODE`

MODE sets the method used to perform RPF lookups. Supported modes:

urib-only Performs the lookup on the Unicast RIB. The Multicast RIB is never used.

mrib-only Performs the lookup on the Multicast RIB. The Unicast RIB is never used.

mrib-then-urib Tries to perform the lookup on the Multicast RIB. If any route is found, that route is used. Otherwise, the Unicast RIB is tried.

lower-distance Performs a lookup on the Multicast RIB and Unicast RIB each. The result with the lower administrative distance is used; if they're equal, the Multicast RIB takes precedence.

longer-prefix Performs a lookup on the Multicast RIB and Unicast RIB each. The result with the longer prefix length is used; if they're equal, the Multicast RIB takes precedence.

The `mrib-then-urib` setting is the default behavior if nothing is configured. If this is the desired behavior, it should be explicitly configured to make the configuration immune against possible changes in what the default behavior is.

Warning: Unreachable routes do not receive special treatment and do not cause fallback to a second lookup.

show [ip|ipv6] rpf ADDR

Performs a Multicast RPF lookup, as configured with `ip multicast rpf-lookup-mode MODE`. ADDR specifies the multicast source address to look up.

```
> show ip rpf 192.0.2.1
Routing entry for 192.0.2.0/24 using Unicast RIB
Known via "kernel", distance 0, metric 0, best
* 198.51.100.1, via eth0
```

Indicates that a multicast source lookup for 192.0.2.1 would use an Unicast RIB entry for 192.0.2.0/24 with a gateway of 198.51.100.1.

show [ip|ipv6] rpf

Prints the entire Multicast RIB. Note that this is independent of the configured RPF lookup mode, the Multicast RIB may be printed yet not used at all.

ip mroute PREFIX NEXTHOP [DISTANCE]

Adds a static route entry to the Multicast RIB. This performs exactly as the `ip route` command, except that it inserts the route in the Multicast RIB instead of the Unicast RIB.

3.1.15 zebra Route Filtering

Zebra supports *prefix-list* s and *Route Maps* s to match routes received from other FRR components. The permit/deny facilities provided by these commands can be used to filter which routes zebra will install in the kernel.

ip protocol PROTOCOL route-map ROUTEMAP

Apply a route-map filter to routes for the specified protocol. PROTOCOL can be:

- any,
- babel,
- bgp,
- connected,
- eigrp,
- isis,
- kernel,
- nhrp,
- openfabric,
- ospf,
- ospf6,
- rip,
- sharp,
- static,
- ripng,
- table,
- vnc.

If you choose any as the option that will cause all protocols that are sending routes to zebra. You can specify a *ip protocol PROTOCOL route-map ROUTEMAP* on a per vrf basis, by entering this command under vrf mode for the vrf you want to apply the route-map against.

set src ADDRESS
Within a route-map, set the preferred source address for matching routes when installing in the kernel.

The following creates a prefix-list that matches all addresses, a route-map that sets the preferred source address, and applies the route-map to all *rip* routes.

```
ip prefix-list ANY permit 0.0.0.0/0 le 32
route-map RM1 permit 10
  match ip address prefix-list ANY
  set src 10.0.0.1

ip protocol rip route-map RM1
```

IPv6 example for OSPFv3.

```
ipv6 prefix-list ANY seq 10 permit any
route-map RM6 permit 10
  match ipv6 address prefix-list ANY
  set src 2001:db8:425:1000::3

ipv6 protocol ospf6 route-map RM6
```

Note: For both IPv4 and IPv6, the IP address has to exist on some interface when the route is getting installed into the system. Otherwise, kernel rejects the route. To solve the problem of disappearing IPv6 addresses when the interface goes down, use `net.ipv6.conf.all.keep_addr_on_down` *sysctl option*.

zebra route-map delay-timer (0-600)
Set the delay before any route-maps are processed in zebra. The default time for this is 5 seconds.

3.1.16 zebra FIB push interface

Zebra supports a 'FIB push' interface that allows an external component to learn the forwarding information computed by the FRR routing suite. This is a loadable module that needs to be enabled at startup as described in *Loadable Module Support*.

In FRR, the Routing Information Base (RIB) resides inside zebra. Routing protocols communicate their best routes to zebra, and zebra computes the best route across protocols for each prefix. This latter information makes up the Forwarding Information Base (FIB). Zebra feeds the FIB to the kernel, which allows the IP stack in the kernel to forward packets according to the routes computed by FRR. The kernel FIB is updated in an OS-specific way. For example, the `Netlink` interface is used on Linux, and route sockets are used on FreeBSD.

The FIB push interface aims to provide a cross-platform mechanism to support scenarios where the router has a forwarding path that is distinct from the kernel, commonly a hardware-based fast path. In these cases, the FIB needs to be maintained reliably in the fast path as well. We refer to the component that programs the forwarding plane (directly or indirectly) as the Forwarding Plane Manager or FPM.

The relevant zebra code kicks in when zebra is configured with the *`--enable-fpm`* flag and started with the module (`-M fpm` or `-M dplane_fpm_nl`).

Note: The `fpm` implementation attempts to connect to `127.0.0.1` port `2620` by default without configurations. The `dplane_fpm_nl` only attempts to connect to a server if configured.

Zebra periodically attempts to connect to the well-known FPM port (`2620`). Once the connection is up, zebra starts sending messages containing routes over the socket to the FPM. Zebra sends a complete copy of the forwarding table to the FPM, including routes that it may have picked up from the kernel. The existing interaction of zebra with the kernel remains unchanged – that is, the kernel continues to receive FIB updates as before.

The default FPM message format is netlink, however it can be controlled with the module load-time option. The modules accept the following options:

- `fpm`: `netlink` and `protobuf`.
- `dplane_fpm_nl`: none, it only implements netlink.

The zebra FPM interface uses replace semantics. That is, if a 'route add' message for a prefix is followed by another 'route add' message, the information in the second message is complete by itself, and replaces the information sent in the first message.

If the connection to the FPM goes down for some reason, zebra sends the FPM a complete copy of the forwarding table(s) when it reconnects.

For more details on the implementation, please read the developer's manual FPM section.

3.1.17 FPM Commands

`fpm` implementation

`fpm connection ip A.B.C.D port (1-65535)`

Configure `zebra` to connect to a different FPM server than the default of `127.0.0.1:2620`

`show zebra fpm stats`

Shows the FPM statistics.

Sample output:

```
Counter                                       Total     Last 10 secs

connect_calls                                     3                2
connect_no_sock                                   0                0
read_cb_calls                                     2                2
write_cb_calls                                    2                0
write_calls                                       1                0
partial_writes                                    0                0
max_writes_hit                                    0                0
t_write_yields                                    0                0
nop_deletes_skipped                               6                0
route_adds                                        5                0
route_dels                                        0                0
updates_triggered                                11                0
redundant_triggers                                0                0
dests_del_after_update                            0                0
t_conn_down_starts                                0                0
t_conn_down_dests_processed                       0                0
t_conn_down_yields                                0                0
```

(continues on next page)

(continued from previous page)

```
t_conn_down_finishes                                 0                0
t_conn_up_starts                                     1                0
t_conn_up_dests_processed                           11                0
t_conn_up_yields                                     0                0
t_conn_up_aborts                                     0                0
t_conn_up_finishes                                   1                0
```

clear zebra fpm stats

Reset statistics related to the zebra code that interacts with the optional Forwarding Plane Manager (FPM) component.

`dplane_fpm_nl` implementation

fpm address <A.B.C.D|X:X::X:X> [port (1-65535)]

Configures the FPM server address. Once configured `zebra` will attempt to connect to it immediately.

The `no` form disables FPM entirely. `zebra` will close any current connections and will not attempt to connect to it anymore.

fpm use-next-hop-groups

Use the new netlink messages `RTM_NEWNEXTHOP` / `RTM_DELNEXTHOP` to group repeated route next hop information.

The `no` form uses the old known FPM behavior of including next hop information in the route (e.g. `RTM_NEWROUTE`) messages.

show fpm counters [json]

Show the FPM statistics (plain text or JSON formatted).

Sample output:

```
                 FPM counters
                 ============
                Input bytes: 0
               Output bytes: 308
 Output buffer current size: 0
    Output buffer peak size: 308
          Connection closes: 0
          Connection errors: 0
 Data plane items processed: 0
  Data plane items enqueued: 0
Data plane items queue peak: 0
           Buffer full hits: 0
    User FPM configurations: 1
  User FPM disable requests: 0
```

clear fpm counters

Reset statistics related to the zebra code that interacts with the optional Forwarding Plane Manager (FPM) component.

3.1.18 Dataplane Commands

The zebra dataplane subsystem provides a framework for FIB programming. Zebra uses the dataplane to program the local kernel as it makes changes to objects such as IP routes, MPLS LSPs, and interface IP addresses. The dataplane runs in its own pthread, in order to off-load work from the main zebra pthread.

`show zebra dplane [detailed]`
: Display statistics about the updates and events passing through the dataplane subsystem.

`show zebra dplane providers`
: Display information about the running dataplane plugins that are providing updates to a FIB. By default, the local kernel plugin is present.

`zebra dplane limit [NUMBER]`
: Configure the limit on the number of pending updates that are waiting to be processed by the dataplane pthread.

3.1.19 DPDK dataplane

The zebra DPDK subsystem programs the dataplane via rte_XXX APIs. This module needs be compiled in via "–enable-dp-dpdk=yes" and enabled at start up time via the zebra daemon option "-M dplane_dpdk".

To program the PBR rules as rte_flows you additionally need to configure "pbr nexthop-resolve". This is used to expland the PBR actions into the {SMAC, DMAC, outgoing port} needed by rte_flow.

`show dplane dpdk port [detail]`
: Displays the mapping table between zebra interfaces and DPDK port-ids. Sample output:

  :: Port Device IfName IfIndex sw,domain,port

  0 0000:03:00.0 p0 4 0000:03:00.0,0,65535 1 0000:03:00.0 pf0hpf 6 0000:03:00.0,0,4095 2 0000:03:00.0 pf0vf0 15 0000:03:00.0,0,4096 3 0000:03:00.0 pf0vf1 16 0000:03:00.0,0,4097 4 0000:03:00.1 p1 5 0000:03:00.1,1,65535 5 0000:03:00.1 pf1hpf 7 0000:03:00.1,1,20479

`show dplane dpdk pbr flows`
`Displays the DPDK stats per-PBR entry.`
`Sample output:`
: :: Rules if pf0vf0 Seq 1 pri 300 SRC Match 77.0.0.8/32 DST Match 88.0.0.8/32 Tableid: 10000 Action: nh: 45.0.0.250 intf: p0 Action: mac: 00:00:5e:00:01:fa DPDK flow: installed 0x40 DPDK flow stats: packets 13 bytes 1586

`show dplane dpdk counters`
`Displays the ZAPI message handler counters`

Sample output:

::

Ignored updates: 0 PBR rule adds: 1 PBR rule dels: 0

3.1.20 zebra Terminal Mode Commands

`show ip route`
Display current routes which zebra holds in its database.

```
Router# show ip route
Codes: K - kernel route, C - connected, S - static, R - RIP,
 B - BGP * - FIB route.

K* 0.0.0.0/0        203.181.89.241
S  0.0.0.0/0        203.181.89.1
C* 127.0.0.0/8      lo
C* 203.181.89.240/28      eth0
```

`show ipv6 route`

`show [ip|ipv6] route [PREFIX] [nexthop-group]`
Display detailed information about a route. If [nexthop-group] is included, it will display the nexthop group ID the route is using as well.

`show interface [NAME] [{vrf VRF|brief}] [json]`

`show interface [NAME] [{vrf all|brief}] [json]`

`show interface [NAME] [{vrf VRF|brief}] [nexthop-group]`

`show interface [NAME] [{vrf all|brief}] [nexthop-group]`
Display interface information. If no extra information is added, it will dump information on all interfaces. If [NAME] is specified, it will display detailed information about that single interface. If [nexthop-group] is specified, it will display nexthop groups pointing out that interface.

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

`show ip prefix-list [NAME]`

`show route-map [NAME]`

`show ip protocol`

`show ip forward`
Display whether the host's IP forwarding function is enabled or not. Almost any UNIX kernel can be configured with IP forwarding disabled. If so, the box can't work as a router.

`show ipv6 forward`
Display whether the host's IP v6 forwarding is enabled or not.

`show ip neigh`
Display the ip neighbor table

`show pbr rule`
Display the pbr rule table with resolved nexthops

`show zebra`
Display various statistics related to the installation and deletion of routes, neighbor updates, and LSP's into the kernel. In addition show various zebra state that is useful when debugging an operator's setup.

`show zebra client [summary]`
Display statistics about clients that are connected to zebra. This is useful for debugging and seeing how much data is being passed between zebra and it's clients. If the summary form of the command is chosen a table is displayed with shortened information.

show zebra router table summary
Display summarized data about tables created, their afi/safi/tableid and how many routes each table contains. Please note this is the total number of route nodes in the table. Which will be higher than the actual number of routes that are held.

show nexthop-group rib [ID] [vrf NAME] [singleton [ip|ip6]] [type] [json]
Display nexthop groups created by zebra. The [vrf NAME] option is only meaningful if you have started zebra with the –vrfwnetns option as that nexthop groups are per namespace in linux. If you specify singleton you would like to see the singleton nexthop groups that do have an afi. [type] allows you to filter those only coming from a specific NHG type (protocol).

show <ip|ipv6> zebra route dump [<vrf> VRFNAME]
It dumps all the routes from RIB with detailed information including internal flags, status etc. This is defined as a hidden command.

3.1.21 Router-id

Many routing protocols require a router-id to be configured. To have a consistent router-id across all daemons, the following commands are available to configure and display the router-id:

[ip] router-id A.B.C.D
Allow entering of the router-id. This command also works under the vrf subnode, to allow router-id's per vrf.

[ip] router-id A.B.C.D vrf NAME
Configure the router-id of this router from the configure NODE. A show run of this command will display the router-id command under the vrf sub node. This command is deprecated and will be removed at some point in time in the future.

show [ip] router-id [vrf NAME]
Display the user configured router-id.

For protocols requiring an IPv6 router-id, the following commands are available:

ipv6 router-id X:X::X:X
Configure the IPv6 router-id of this router. Like its IPv4 counterpart, this command works under the vrf subnode, to allow router-id's per vrf.

show ipv6 router-id [vrf NAME]
Display the user configured IPv6 router-id.

3.1.22 sysctl settings

The linux kernel has a variety of sysctl's that affect it's operation as a router. This section is meant to act as a starting point for those sysctl's that must be used in order to provide FRR with smooth operation as a router. This section is not meant as the full documentation for sysctl's. The operator must use the sysctl documentation with the linux kernel for that. The following link has helpful references to many relevant sysctl values: https://www.kernel.org/doc/Documentation/networking/ip-sysctl.txt

Expected sysctl settings

`net.ipv4.ip_forward = 1`
: This global option allows the linux kernel to forward (route) ipv4 packets incoming from one interface to an outgoing interface. If this is set to 0, the system will not route transit ipv4 packets, i.e. packets that are not sent to/from a process running on the local system.

`net.ipv4.conf.{all,default,<interface>}.forwarding = 1`
: The linux kernel can selectively enable forwarding (routing) of ipv4 packets on a per interface basis. The forwarding check in the kernel dataplane occurs against the ingress Layer 3 interface, i.e. if the ingress L3 interface has forwarding set to 0, packets will not be routed.

`net.ipv6.conf.{all,default,<interface>}.forwarding = 1`
: This per interface option allows the linux kernel to forward (route) transit ipv6 packets i.e. incoming from one Layer 3 interface to an outgoing Layer 3 interface. The forwarding check in the kernel dataplane occurs against the ingress Layer 3 interface, i.e. if the ingress L3 interface has forwarding set to 0, packets will not be routed.

`net.ipv6.conf.all.keep_addr_on_down = 1`
: When an interface is taken down, do not remove the v6 addresses associated with the interface. This option is recommended because this is the default behavior for v4 as well.

`net.ipv6.route.skip_notify_on_dev_down = 1`
: When an interface is taken down, the linux kernel will not notify, via netlink, about routes that used that interface being removed from the FIB. This option is recommended because this is the default behavior for v4 as well.

Optional sysctl settings

`net.ipv4.conf.{all,default,<interface>}.bc_forwarding = 0`
: This per interface option allows the linux kernel to optionally allow Directed Broadcast (i.e. Routed Broadcast or Subnet Broadcast) packets to be routed onto the connected network segment where the subnet exists. If the local router receives a routed packet destined for a broadcast address of a connected subnet, setting bc_forwarding to 1 on the interface with the target subnet assigned to it will allow non locally-generated packets to be routed via the broadcast route. If bc_forwarding is set to 0, routed packets destined for a broadcast route will be dropped. e.g. Host1 (SIP:192.0.2.10, DIP:10.0.0.255) -> (eth0:192.0.2.1/24) Router1 (eth1:10.0.0.1/24) -> BC If net.ipv4.conf.{all,default,<interface>}.bc_forwarding=1, then Router1 will forward each packet destined to 10.0.0.255 onto the eth1 interface with a broadcast DMAC (ff:ff:ff:ff:ff:ff).

`net.ipv4.conf.{all,default,<interface>}.arp_accept = 1`
: This per interface option allows the linux kernel to optionally skip the creation of ARP entries upon the receipt of a Gratuitous ARP (GARP) frame carrying an IP that is not already present in the ARP cache. Setting arp_accept to 0 on an interface will ensure NEW ARP entries are not created due to the arrival of a GARP frame. Note: This does not impact how the kernel reacts to GARP frames that carry a "known" IP (that is already in the ARP cache) – an existing ARP entry will always be updated when a GARP for that IP is received.

`net.ipv4.conf.{all,default,<interface>}.arp_ignore = 0`
: This per interface option allows the linux kernel to control what conditions must be met in order for an ARP reply to be sent in response to an ARP request targeting a local IP address. When arp_ignore is set to 0, the kernel will send ARP replies in response to any ARP Request with a Target-IP matching a local address. When arp_ignore is set to 1, the kernel will send ARP replies if the Target-IP in the ARP Request matches an IP address on the interface the Request arrived at. When arp_ignore is set to 2, the kernel will send ARP replies only if the Target-IP matches an IP address on the interface where the Request arrived AND the Sender-IP falls within the subnet assigned to the local IP/interface.

`net.ipv4.conf.{all,default,<interface>}.arp_notify = 1`
: This per interface option allows the linux kernel to decide whether to send a Gratuitious ARP (GARP) frame when the Layer 3 interface comes UP. When arp_notify is set to 0, no GARP is sent. When arp_notify is set to 1, a GARP is sent when the interface comes UP.

`net.ipv6.conf.{all,default,<interface>}.ndisc_notify = 1`
: This per interface option allows the linux kernel to decide whether to send an Unsolicited Neighbor Advertisement (U-NA) frame when the Layer 3 interface comes UP. When ndisc_notify is set to 0, no U-NA is sent. When ndisc_notify is set to 1, a U-NA is sent when the interface comes UP.

Useful sysctl settings

`net.ipv6.conf.all.use_oif_addrs_only = 1`
: When enabled, the candidate source addresses for destinations routed via this interface are restricted to the set of addresses configured on this interface (RFC 6724 section 4). If an operator has hundreds of IP addresses per interface this solves the latency problem.

3.1.23 Debugging

`debug zebra mpls [detailed]`
: MPLS-related events and information.

`debug zebra events`
: Zebra events

`debug zebra nht [detailed]`
: Nexthop-tracking / reachability information

`debug zebra vxlan`
: VxLAN (EVPN) events

`debug zebra pseudowires`
: Pseudowire events.

`debug zebra packet [<recv|send>] [detail]`
: ZAPI message and packet details

`debug zebra kernel`
: Kernel / OS events.

`debug zebra kernel msgdump [<recv|send>]`
: Raw OS (netlink) message details.

`debug zebra rib [detailed]`
: RIB events.

`debug zebra fpm`
: FPM (forwarding-plane manager) events.

`debug zebra dplane [detailed]`
: Dataplane / FIB events.

`debug zebra pbr`
: PBR (policy-based routing) events.

`debug zebra mlag`
: MLAG events.

`debug zebra evpn mh <es|mac|neigh|nh>`
: EVPN multi-hop events.

`debug zebra nexthop [detail]`
: Nexthop and nexthop-group events.

3.1.24 Scripting

`zebra on-rib-process script SCRIPT`

Set a Lua script for *on_rib_process_dplane_results* hook call. SCRIPT is the basename of the script, without `.lua`.

Data structures

const struct zebra_dplane_ctx

```
* integer zd_op
* integer zd_status
* integer zd_provider
* integer zd_vrf_id
* integer zd_table_id
* integer zd_ifname
* integer zd_ifindex
* table rinfo (if zd_op is DPLANE_OP_ROUTE*, DPLANE_NH_*)

  * prefix zd_dest
  * prefix zd_src
  * integer zd_afi
  * integer zd_safi
  * integer zd_type
  * integer zd_old_type
  * integer zd_tag
  * integer zd_old_tag
  * integer zd_metric
  * integer zd_old_metric
  * integer zd_instance
  * integer zd_old_instance
  * integer zd_distance
  * integer zd_old_distance
  * integer zd_mtu
  * integer zd_nexthop_mtu
  * table nhe

    * integer id
    * integer old_id
    * integer afi
    * integer vrf_id
    * integer type
    * nexthop_group ng
    * nh_grp
    * integer nh_grp_count

  * integer zd_nhg_id
  * nexthop_group zd_ng
  * nexthop_group backup_ng
  * nexthop_group zd_old_ng
  * nexthop_group old_backup_ng
```

(continues on next page)

(continued from previous page)

```
* integer label (if zd_op is DPLANE_OP_LSP_*)
* table pw (if zd_op is DPLANE_OP_PW_*)

  * integer type
  * integer af
  * integer status
  * integer flags
  * integer local_label
  * integer remote_label

* table macinfo (if zd_op is DPLANE_OP_MAC_*)

  * integer vid
  * integer br_ifindex
  * ethaddr mac
  * integer vtep_ip
  * integer is_sticky
  * integer nhg_id
  * integer update_flags

* table rule (if zd_op is DPLANE_OP_RULE_*)

  * integer sock
  * integer unique
  * integer seq
  * string ifname
  * integer priority
  * integer old_priority
  * integer table
  * integer old_table
  * integer filter_bm
  * integer old_filter_bm
  * integer fwmark
  * integer old_fwmark
  * integer dsfield
  * integer old_dsfield
  * integer ip_proto
  * integer old_ip_proto
  * prefix src_ip
  * prefix old_src_ip
  * prefix dst_ip
  * prefix old_dst_ip

* table iptable (if zd_op is DPLANE_OP_IPTABLE_*)

  * integer sock
  * integer vrf_id
  * integer unique
  * integer type
  * integer filter_bm
  * integer fwmark
  * integer action
```

(continues on next page)

(continued from previous page)

```
  * integer pkt_len_min
  * integer pkt_len_max
  * integer tcp_flags
  * integer dscp_value
  * integer fragment
  * integer protocol
  * integer nb_interface
  * integer flow_label
  * integer family
  * string ipset_name

* table ipset (if zd_op is DPLANE_OP_IPSET_*)
  * integer sock
  * integer vrf_id
  * integer unique
  * integer type
  * integer family
  * string ipset_name

* table neigh (if zd_op is DPLANE_OP_NEIGH_*)

  * ipaddr ip_addr
  * table link

    * ethaddr mac
    * ipaddr ip_addr

  * integer flags
  * integer state
  * integer update_flags

* table br_port (if zd_op is DPLANE_OP_BR_PORT_UPDATE)

  * integer sph_filter_cnt
  * integer flags
  * integer backup_nhg_id

* table neightable (if zd_op is DPLANE_OP_NEIGH_TABLE_UPDATE)

  * integer family
  * integer app_probes
  * integer ucast_probes
  * integer mcast_probes

* table gre (if zd_op is DPLANE_OP_GRE_SET)**

  * integer link_ifindex
  * integer mtu
```

const struct nh_grp

```
* integer id
* integer weight
```

Zebra Hook calls

on_rib_process_dplane_results

Called when RIB processes dataplane events. Set script location with the `zebra on-rib-process script SCRIPT` command.

Arguments

- *const struct zebra_dplane_ctx* ctx

```
function on_rib_process_dplane_results(ctx)
   log.info(ctx.rinfo.zd_dest.network)
   return {}
```

3.2 Bidirectional Forwarding Detection

BFD (Bidirectional Forwarding Detection) stands for Bidirectional Forwarding Detection and it is described and extended by the following RFCs:

- **RFC 5880**
- **RFC 5881**
- **RFC 5882**
- **RFC 5883**

Currently, there are two implementations of the BFD commands in FRR:

- PTM (Prescriptive Topology Manager): an external daemon which implements BFD;
- `bfdd`: a BFD implementation that is able to talk with remote peers;

This document will focus on the later implementation: *bfdd*.

3.2.1 Starting BFD

bfdd default configuration file is `bfdd.conf`. *bfdd* searches the current directory first then /etc/frr/bfdd.conf. All of *bfdd*'s command must be configured in `bfdd.conf`.

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

`--bfdctl` `<unix-socket>`

Set the BFD daemon control socket location. If using a non-default socket location:

```
/usr/lib/frr/bfdd --bfdctl /tmp/bfdd.sock
```

The default UNIX socket location is:

> #define BFDD_CONTROL_SOCKET "/var/run/frr/bfdd.sock"

This option overrides the location addition that the -N option provides to the bfdd.sock

`--dplaneaddr` `<type>:<address>[<:port>]`

Configure the distributed BFD data plane listening socket bind address.

One would expect the data plane to run in the same machine as FRR, so the suggested configuration would be:

> –dplaneaddr unix:/var/run/frr/bfdd_dplane.sock

Or using IPv4:

> –dplaneaddr ipv4:127.0.0.1

Or using IPv6:

> –dplaneaddr ipv6:[::1]

It is also possible to specify a port (for IPv4/IPv6 only):

> –dplaneaddr ipv6:[::1]:50701

(if ommited the default port is `50700`).

It is also possible to operate in client mode (instead of listening for connections). To connect to a data plane server append the letter 'c' to the protocol, example:

> –dplaneaddr ipv4c:127.0.0.1

Note: When using UNIX sockets don't forget to check the file permissions before attempting to use it.

3.2.2 BFDd Commands

`bfd`

Opens the BFD daemon configuration node.

`peer <A.B.C.D|X:X::X:X> [{multihop|local-address <A.B.C.D|X:X::X:X>|interface IFNAME|vrf NAME}]`

Creates and configures a new BFD peer to listen and talk to.

multihop tells the BFD daemon that we should expect packets with TTL less than 254 (because it will take more than one hop) and to listen on the multihop port (4784). When using multi-hop mode *echo-mode* will not work (see **RFC 5883** section 3).

local-address provides a local address that we should bind our peer listener to and the address we should use to send the packets. This option is mandatory for IPv6.

interface selects which interface we should use.

vrf selects which domain we want to use.

`profile WORD`

Creates a peer profile that can be configured in multiple peers.

Deleting the profile will cause all peers using it to reset to the default values.

`show bfd [vrf NAME] peers [json]`

Show all configured BFD peers information and current status.

`show bfd [vrf NAME] peer <WORD|<A.B.C.D|X:X::X:X> [{multihop|local-address <A.B.C.D|X:X::X:X>|interface IFNAME}]> [json]`

Show status for a specific BFD peer.

show bfd [vrf NAME] peers brief [json]
Show all configured BFD peers information and current status in brief.

show bfd distributed
Show the BFD data plane (distributed BFD) statistics.

Peer / Profile Configuration

BFD peers and profiles share the same BFD session configuration commands.

detect-multiplier (2-255)
Configures the detection multiplier to determine packet loss. The remote transmission interval will be multiplied by this value to determine the connection loss detection timer. The default value is 3.

Example: when the local system has *detect-multiplier 3* and the remote system has *transmission interval 300*, the local system will detect failures only after 900 milliseconds without receiving packets.

receive-interval (10-60000)
Configures the minimum interval that this system is capable of receiving control packets. The default value is 300 milliseconds.

transmit-interval (10-60000)
The minimum transmission interval (less jitter) that this system wants to use to send BFD control packets. Defaults to 300ms.

echo receive-interval <disabled|(10-60000)>
Configures the minimum interval that this system is capable of receiving echo packets. Disabled means that this system doesn't want to receive echo packets. The default value is 50 milliseconds.

echo transmit-interval (10-60000)
The minimum transmission interval (less jitter) that this system wants to use to send BFD echo packets. Defaults to 50ms.

echo-mode
Enables or disables the echo transmission mode. This mode is disabled by default. If you are not using distributed BFD then echo mode works only when the peer is also FRR.

It is recommended that the transmission interval of control packets to be increased after enabling echo-mode to reduce bandwidth usage. For example: *transmit-interval 2000*.

Echo mode is not supported on multi-hop setups (see **RFC 5883** section 3).

shutdown
Enables or disables the peer. When the peer is disabled an 'administrative down' message is sent to the remote peer.

passive-mode
Mark session as passive: a passive session will not attempt to start the connection and will wait for control packets from peer before it begins replying.

This feature is useful when you have a router that acts as the central node of a star network and you want to avoid sending BFD control packets you don't need to.

The default is active-mode (or `no passive-mode`).

minimum-ttl (1-254)
For multi hop sessions only: configure the minimum expected TTL for an incoming BFD control packet.

This feature serves the purpose of thightening the packet validation requirements to avoid receiving BFD control packets from other sessions.

The default value is 254 (which means we only expect one hop between this system and the peer).

BFD Peer Specific Commands

`label WORD`
Labels a peer with the provided word. This word can be referenced later on other daemons to refer to a specific peer.

`profile BFDPROF`
Configure peer to use the profile configurations.

Notes:

- Profile configurations can be overridden on a peer basis by specifying non-default parameters in peer configuration node.
- Non existing profiles can be configured and they will only be applied once they start to exist.
- If the profile gets updated the new configuration will be applied to all peers with the profile without interruptions.

BGP BFD Configuration

The following commands are available inside the BGP configuration node.

`neighbor <A.B.C.D|X:X::X:X|WORD> bfd`
Listen for BFD events registered on the same target as this BGP neighbor. When BFD peer goes down it immediately asks BGP to shutdown the connection with its neighbor and, when it goes back up, notify BGP to try to connect to it.

`neighbor <A.B.C.D|X:X::X:X|WORD> bfd check-control-plane-failure`
Allow to write CBIT independence in BFD outgoing packets. Also allow to read both C-BIT value of BFD and lookup BGP peer status. This command is useful when a BFD down event is caught, while the BGP peer requested that local BGP keeps the remote BGP entries as staled if such issue is detected. This is the case when graceful restart is enabled, and it is wished to ignore the BD event while waiting for the remote router to restart.

Disabling this disables presence of CBIT independence in BFD outgoing packets and pays attention to BFD down notifications. This is the default.

`neighbor <A.B.C.D|X:X::X:X|WORD> bfd profile BFDPROF`
Same as command `neighbor <A.B.C.D|X:X::X:X|WORD> bfd`, but applies the BFD profile to the sessions it creates or that already exist.

IS-IS BFD Configuration

The following commands are available inside the interface configuration node.

`isis bfd`
Listen for BFD events on peers created on the interface. Every time a new neighbor is found a BFD peer is created to monitor the link status for fast convergence.

Note that there will be just one BFD session per interface. In case both IPv4 and IPv6 support are configured then just a IPv6 based session is created.

`isis bfd profile BFDPROF`
Use a BFD profile BFDPROF as provided in the BFD configuration.

OSPF BFD Configuration

The following commands are available inside the interface configuration node.

`ip ospf bfd`

Listen for BFD events on peers created on the interface. Every time a new neighbor is found a BFD peer is created to monitor the link status for fast convergence.

`ip ospf bfd profile BFDPROF`

Same as command `ip ospf bfd`, but applies the BFD profile to the sessions it creates or that already exist.

OSPF6 BFD Configuration

The following commands are available inside the interface configuration node.

`ipv6 ospf6 bfd [profile BFDPROF]`

Listen for BFD events on peers created on the interface. Every time a new neighbor is found a BFD peer is created to monitor the link status for fast convergence.

Optionally uses the BFD profile `BFDPROF` in the created sessions under that interface.

PIM BFD Configuration

The following commands are available inside the interface configuration node.

`ip pim bfd [profile BFDPROF]`

Listen for BFD events on peers created on the interface. Every time a new neighbor is found a BFD peer is created to monitor the link status for fast convergence.

Optionally uses the BFD profile `BFDPROF` in the created sessions under that interface.

RIP BFD configuration

The following commands are available inside the interface configuration node:

`ip rip bfd`

Automatically create BFD session for each RIP peer discovered in this interface. When the BFD session monitor signalize that the link is down the RIP peer is removed and all the learned routes associated with that peer are removed.

`ip rip bfd profile BFD_PROFILE_NAME`

Selects a BFD profile for the BFD sessions created in this interface.

The following command is available in the RIP router configuration node:

`bfd default-profile BFD_PROFILE_NAME`

Selects a default BFD profile for all sessions without a profile specified.

BFD Static Route Monitoring Configuration

A monitored static route conditions the installation to the RIB on the BFD session running state: when BFD session is up the route is installed to RIB, but when the BFD session is down it is removed from the RIB.

The following commands are available inside the configuration node:

`ip route A.B.C.D/M A.B.C.D bfd [{multi-hop|source A.B.C.D|profile BFDPROF}]`

Configure a static route for `A.B.C.D/M` using gateway `A.B.C.D` and use the gateway address as BFD peer destination address.

`ipv6 route X:X::X:X/M [from X:X::X:X/M] X:X::X:X bfd [{multi-hop|source X:X::X:X|profile BFDPROF}]`

Configure a static route for `X:X::X:X/M` using gateway `X:X::X:X` and use the gateway address as BFD peer destination address.

The static routes when uninstalled will no longer show up in the output of the command `show ip route` or `show ipv6 route`, instead we must use the BFD static route show command to see these monitored route status.

`show bfd static route [json]`

Show all monitored static routes and their status.

Example output:

```
Showing BFD monitored static routes:

  Route groups:
    rtg1 peer 172.16.0.1 (status: uninstalled):
        2001:db8::100/128

Next hops:
  VRF default IPv4 Unicast:
      192.168.100.0/24 peer 172.16.0.1 (status: uninstalled)

  VRF default IPv4 Multicast:

  VRF default IPv6 Unicast:
```

3.2.3 Configuration

Before applying `bfdd` rules to integrated daemons (like BGPd), we must create the corresponding peers inside the `bfd` configuration node.

Here is an example of BFD configuration:

```
bfd
 peer 192.168.0.1
   label home-peer
   no shutdown
 !
!
router bgp 65530
 neighbor 192.168.0.1 remote-as 65531
 neighbor 192.168.0.1 bfd
 neighbor 192.168.0.2 remote-as 65530
 neighbor 192.168.0.2 bfd
```

(continues on next page)

(continued from previous page)

```
 neighbor 192.168.0.3 remote-as 65532
 neighbor 192.168.0.3 bfd
!
```

Peers can be identified by its address (use `multihop` when you need to specify a multi hop peer) or can be specified manually by a label.

Here are the available peer configurations:

```
bfd
 ! Configure a fast profile
 profile fast
  receive-interval 150
  transmit-interval 150
 !

 ! Configure peer with fast profile
 peer 192.168.0.6
  profile fast
  no shutdown
 !

! Configure peer with fast profile and override receive speed.
 peer 192.168.0.7
  profile fast
  receive-interval 500
  no shutdown
 !

 ! configure a peer on an specific interface
 peer 192.168.0.1 interface eth0
  no shutdown
 !

 ! configure a multihop peer
 peer 192.168.0.2 multihop local-address 192.168.0.3
   shutdown
 !

 ! configure a peer in a different vrf
 peer 192.168.0.3 vrf foo
  shutdown
 !

 ! configure a peer with every option possible
 peer 192.168.0.4
  label peer-label
  detect-multiplier 50
  receive-interval 60000
  transmit-interval 3000
  shutdown
 !
```

(continues on next page)

(continued from previous page)

```
! configure a peer on an interface from a separate vrf
peer 192.168.0.5 interface eth1 vrf vrf2
 no shutdown
!

! remove a peer
no peer 192.168.0.3 vrf foo
```

3.2.4 Status

You can inspect the current BFD peer status with the following commands:

```
frr# show bfd peers
BFD Peers:
        peer 192.168.0.1
                ID: 1
                Remote ID: 1
                Status: up
                Uptime: 1 minute(s), 51 second(s)
                Diagnostics: ok
                Remote diagnostics: ok
                Peer Type: dynamic
                Local timers:
                        Detect-multiplier: 3
                        Receive interval: 300ms
                        Transmission interval: 300ms
                        Echo receive interval: 50ms
                        Echo transmission interval: disabled
                Remote timers:
                        Detect-multiplier: 3
                        Receive interval: 300ms
                        Transmission interval: 300ms
                        Echo receive interval: 50ms

        peer 192.168.1.1
                label: router3-peer
                ID: 2
                Remote ID: 2
                Status: up
                Uptime: 1 minute(s), 53 second(s)
                Diagnostics: ok
                Remote diagnostics: ok
                Peer Type: configured
                Local timers:
                        Detect-multiplier: 3
                        Receive interval: 300ms
                        Transmission interval: 300ms
                        Echo receive interval: 50ms
                        Echo transmission interval: disabled
                Remote timers:
                        Detect-multiplier: 3
```

(continues on next page)

(continued from previous page)

```
                        Receive interval: 300ms
                        Transmission interval: 300ms
                        Echo receive interval: 50ms

frr# show bfd peer 192.168.1.1
BFD Peer:
            peer 192.168.1.1
                label: router3-peer
                ID: 2
                Remote ID: 2
                Status: up
                Uptime: 3 minute(s), 4 second(s)
                Diagnostics: ok
                Remote diagnostics: ok
                Peer Type: dynamic
                Local timers:
                        Detect-multiplier: 3
                        Receive interval: 300ms
                        Transmission interval: 300ms
                        Echo receive interval: 50ms
                        Echo transmission interval: disabled
                Remote timers:
                        Detect-multiplier: 3
                        Receive interval: 300ms
                        Transmission interval: 300ms
                        Echo receive interval: 50ms

frr# show bfd peer 192.168.0.1 json
{"multihop":false,"peer":"192.168.0.1","id":1,"remote-id":1,"status":"up","uptime":161,
↪"diagnostic":"ok","remote-diagnostic":"ok","receive-interval":300,"transmit-interval
↪":300,"echo-receive-interval":50,"echo-transmit-interval":0,"detect-multiplier":3,
↪"remote-receive-interval":300,"remote-transmit-interval":300,"remote-echo-receive-
↪interval":50,"remote-detect-multiplier":3,"peer-type":"dynamic"}
```

If you are running IPV4 BFD Echo, on a Linux platform, we also calculate round trip time for the packets. We display minimum, average and maximum time it took to receive the looped Echo packets in the RTT fields.

You can inspect the current BFD peer status in brief with the following commands:

```
frr# show bfd peers brief
Session count: 1
SessionId  LocalAddress          PeerAddress      Status
=========  ============          ===========      ======
1          192.168.0.1           192.168.0.2      up
```

You can also inspect peer session counters with the following commands:

```
frr# show bfd peers counters
BFD Peers:
     peer 192.168.2.1 interface r2-eth2
             Control packet input: 28 packets
             Control packet output: 28 packets
             Echo packet input: 0 packets
```

(continues on next page)

(continued from previous page)

```
                Echo packet output: 0 packets
                Session up events: 1
                Session down events: 0
                Zebra notifications: 2

        peer 192.168.0.1
                Control packet input: 54 packets
                Control packet output: 103 packets
                Echo packet input: 965 packets
                Echo packet output: 966 packets
                Session up events: 1
                Session down events: 0
                Zebra notifications: 4

frr# show bfd peer 192.168.0.1 counters
        peer 192.168.0.1
                Control packet input: 126 packets
                Control packet output: 247 packets
                Echo packet input: 2409 packets
                Echo packet output: 2410 packets
                Session up events: 1
                Session down events: 0
                Zebra notifications: 4

frr# show bfd peer 192.168.0.1 counters json
{"multihop":false,"peer":"192.168.0.1","control-packet-input":348,"control-packet-output
↪":685,"echo-packet-input":6815,"echo-packet-output":6816,"session-up":1,"session-down
↪":0,"zebra-notifications":4}
```

You can also clear packet counters per session with the following commands, only the packet counters will be reset:

```
frr# clear bfd peers counters

frr# show bfd peers counters
BFD Peers:
        peer 192.168.2.1 interface r2-eth2
                Control packet input: 0 packets
                Control packet output: 0 packets
                Echo packet input: 0 packets
                Echo packet output: 0 packets
                Session up events: 1
                Session down events: 0
                Zebra notifications: 2

        peer 192.168.0.1
                Control packet input: 0 packets
                Control packet output: 0 packets
                Echo packet input: 0 packets
                Echo packet output: 0 packets
                Session up events: 1
                Session down events: 0
                Zebra notifications: 4
```

3.2.5 Distributed BFD

The distributed BFD is the separation of the BFD protocol control plane from the data plane. FRR implements its own BFD data plane protocol so vendors can study and include it in their own software/hardware without having to modify the FRR source code. The protocol definitions can be found at `bfdd/bfddp_packet.h` header (or the installed `/usr/include/frr/bfdd/bfddp_packet.h`).

To use this feature the BFD daemon needs to be started using the command line option *--dplaneaddr*. When operating using this option the BFD daemon will not attempt to establish BFD sessions, but it will offload all its work to the data plane that is (or will be) connected. Data plane reconnection is also supported.

The BFD data plane will be responsible for:

- Sending/receiving the BFD protocol control/echo packets
- Notifying BFD sessions state changes
- Keeping the number of packets/bytes received/transmitted per session

The FRR BFD daemon will be responsible for:

- Adding/updating BFD session settings
- Asking for BFD session counters
- Redistributing the state changes to the integrated protocols (`bgpd`, `ospfd` etc…)

BFD daemon will also keep record of data plane communication statistics with the command *show bfd distributed*.

Sample output:

```
frr# show bfd distributed
            Data plane
            ==========
       File descriptor: 16
           Input bytes: 1296
      Input bytes peak: 72
        Input messages: 42
   Input current usage: 0
          Output bytes: 568
     Output bytes peak: 136
       Output messages: 19
    Output full events: 0
  Output current usage: 0
```

3.2.6 Debugging

By default only informational, warning and errors messages are going to be displayed. If you want to get debug messages and other diagnostics then make sure you have *debugging* level enabled:

```
config
log file /var/log/frr/frr.log debugging
log syslog debugging
```

You may also fine tune the debug messages by selecting one or more of the debug levels:

`debug bfd distributed`
Toggle BFD data plane (distributed BFD) debugging.

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.

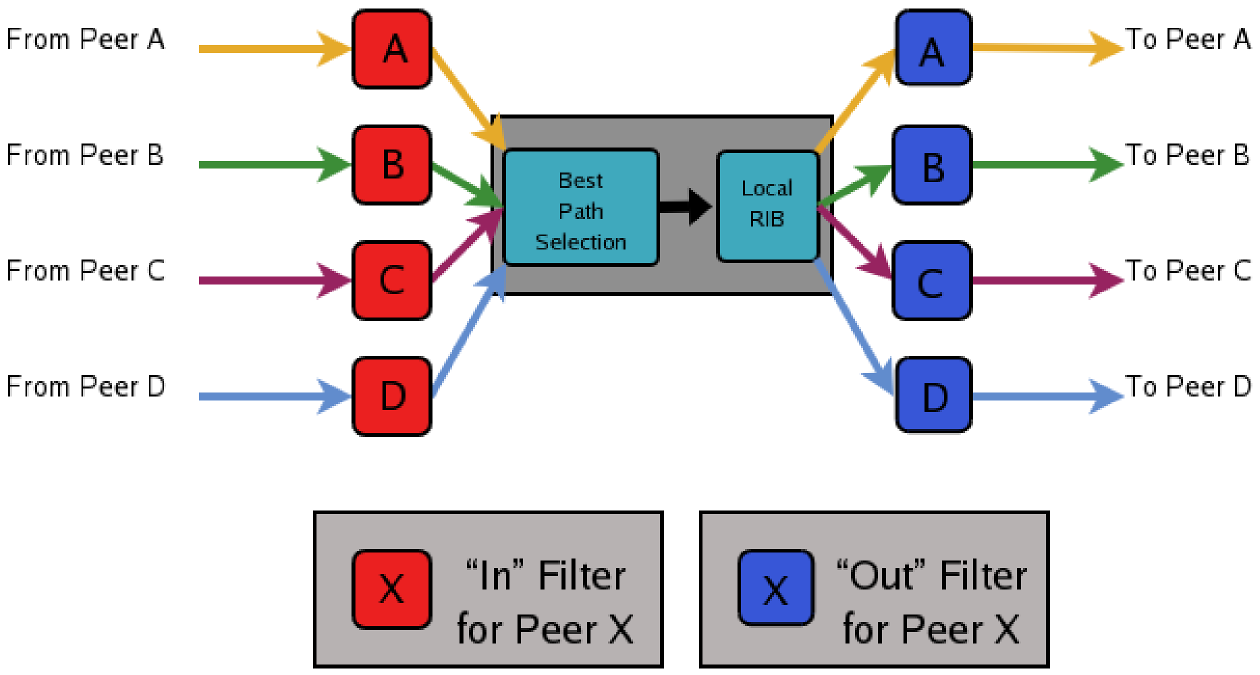


Fig. 1: Announcement processing inside a 'normal' BGP speaker

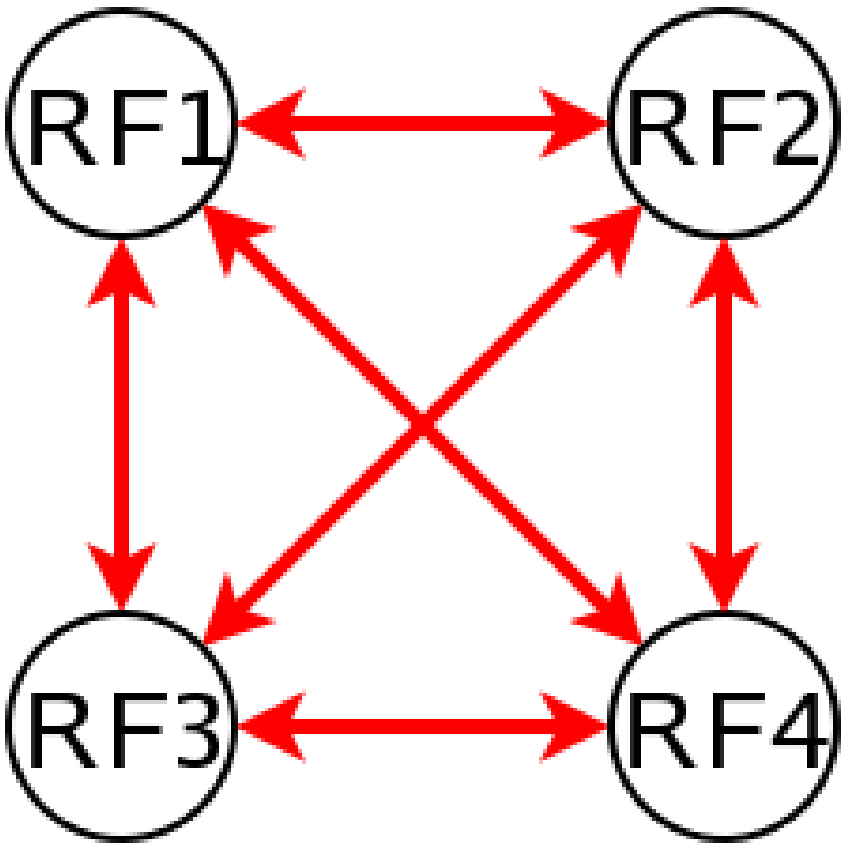


Fig. 2: Full Mesh

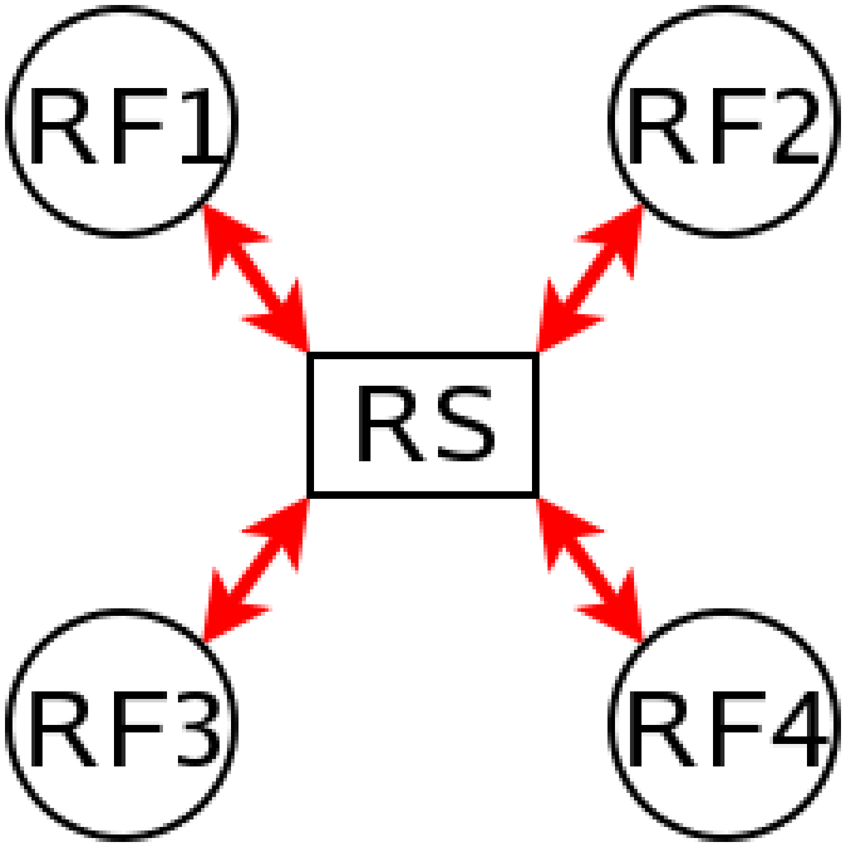


Fig. 3: Route server and clients

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).

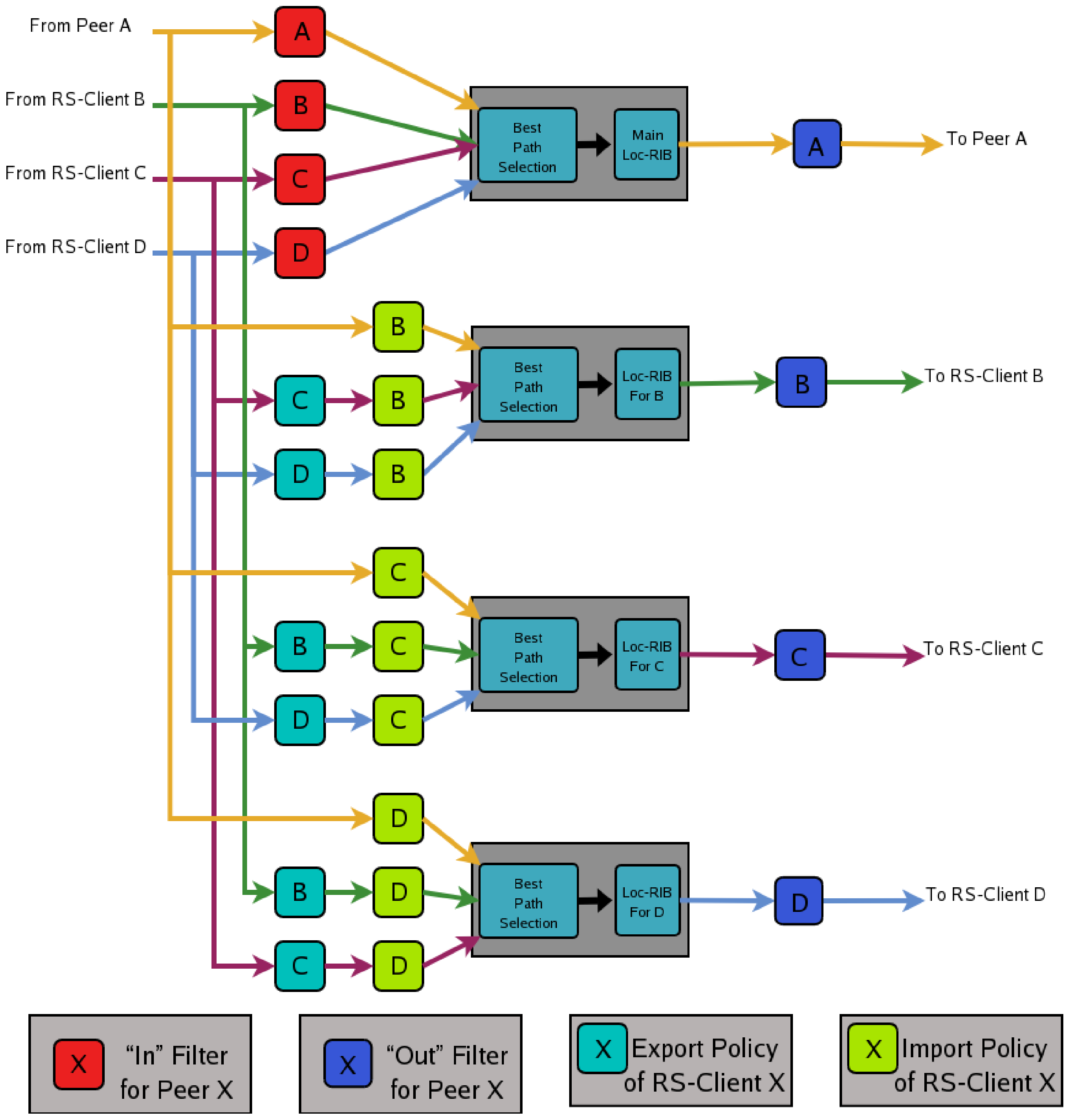


Fig. 4: Announcement processing model implemented by the Route Server

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 are mapped on Linux Network Namespaces. For data traffic to cross VRF boundaries, virtual ethernet interfaces are created with private IP addressing scheme.

```
router bgp <ASx>
 neighbor <A.B.C.D> remote-as <ASz>
 address-family ipv4 flowspec
  neighbor A.B.C.D activate
 exit
exit
router bgp <ASy> vrf vrf2
 address-family ipv4 unicast
  rt redirect import <E.F.G.H:II>
 exit
exit
```

Similarly, it is possible to do the same for IPv6 flowspec rules, by using an IPv6 extended community. The format is defined on **RFC 5701**, and that community contains an IPv6 address encoded in the attribute, and matches the locally configured imported route target IPv6 defined under the appropriate BGP VRF instance. Below example defines an IPv6 extended community containing *E:F::G:H* address followed by 2 bytes chosen by admin (here *JJ*).

```
router bgp <ASx>
 neighbor <A:B::C:D> remote-as <ASz>
 address-family ipv6 flowspec
  neighbor A:B::C:D activate
 exit
exit
router bgp <ASy> vrf vrf2
 address-family ipv6 unicast
  rt6 redirect import <E:F::G:H:JJ>
 exit
exit
```

Flowspec monitoring & troubleshooting

You can monitor policy-routing objects by using one of the following commands. Those command rely on the filtering contexts configured from BGP, and get the statistics information retrieved from the underlying system. In other words, those statistics are retrieved from `Netfilter`.

`show pbr ipset IPSETNAME | iptable`

`IPSETNAME` is the policy routing object name created by `ipset`. About rule contexts, it is possible to know which rule has been configured to policy-route some specific traffic. The `show pbr iptable` command displays for forwarded traffic, which table is used. Then it is easy to use that table identifier to dump the routing table that the forwarded traffic will match.

```

```

`show ip route table TABLEID`

`TABLEID` is the table number identifier referencing the non standard routing table used in this example.

`debug bgp flowspec`

You can troubleshoot Flowspec, or BGP policy based routing. For instance, if you encounter some issues when decoding a Flowspec entry, you should enable *`debug bgp flowspec`*.

debug bgp pbr [error]

If you fail to apply the flowspec entry into *zebra*, there should be some relationship with policy routing mechanism. Here, `debug bgp pbr error` could help.

To get information about policy routing contexts created/removed, only use `debug bgp pbr` command.

Ensuring that a Flowspec entry has been correctly installed and that incoming traffic is policy-routed correctly can be checked as demonstrated below. First of all, you must check whether the Flowspec entry has been installed or not.

```
CLI# show bgp ipv4 flowspec 5.5.5.2/32
 BGP flowspec entry: (flags 0x418)
   Destination Address 5.5.5.2/32
   IP Protocol = 17
   Destination Port >= 50 , <= 90
   FS:redirect VRF RT:255.255.255.255:255
   received for 18:41:37
   installed in PBR (match0x271ce00)
```

This means that the Flowspec entry has been installed in an `iptable` named `match0x271ce00`. Once you have confirmation it is installed, you can check whether you find the associate entry by executing following command. You can also check whether incoming traffic has been matched by looking at counter line.

```
CLI# show pbr ipset match0x271ce00
IPset match0x271ce00 type net,port
     to 5.5.5.0/24:proto 6:80-120 (8)
        pkts 1000, bytes 1000000
     to 5.5.5.2:proto 17:50-90 (5)
        pkts 1692918, bytes 157441374
```

As you can see, the entry is present. note that an `iptable` entry can be used to host several Flowspec entries. In order to know where the matching traffic is redirected to, you have to look at the policy routing rules. The policy-routing is done by forwarding traffic to a routing table number. That routing table number is reached by using a `iptable`. The relationship between the routing table number and the incoming traffic is a `MARKER` that is set by the IPtable referencing the IPSet. In Flowspec case, `iptable` referencing the `ipset` context have the same name. So it is easy to know which routing table is used by issuing following command:

```
CLI# show pbr iptable
   IPtable match0x271ce00 action redirect (5)
     pkts 1700000, bytes 158000000
     table 257, fwmark 257
...
```

As you can see, by using following Linux commands, the MARKER `0x101` is present in both `iptable` and `ip rule` contexts.

```
# iptables -t mangle --list match0x271ce00 -v
Chain match0x271ce00 (1 references)
pkts bytes target     prot opt in     out     source               destination
1700K  158M MARK       all  --  any    any     anywhere             anywhere
     MARK set 0x101
1700K  158M ACCEPT     all  --  any    any     anywhere             anywhere

# ip rule list
0:from all lookup local
0:from all fwmark 0x101 lookup 257
```

(continues on next page)

(continued from previous page)

```
32766:from all lookup main
32767:from all lookup default
```

This allows us to see where the traffic is forwarded to.

Limitations / Known Issues

As you can see, Flowspec is rich and can be very complex. As of today, not all Flowspec rules will be able to be converted into Policy Based Routing actions.

- The `Netfilter` driver is not integrated into FRR yet. Not having this piece of code prevents from injecting flowspec entries into the underlying system.
- There are some limitations around filtering contexts

  If I take example of UDP ports, or TCP ports in Flowspec, the information can be a range of ports, or a unique value. This case is handled. However, complexity can be increased, if the flow is a combination of a list of range of ports and an enumerate of unique values. Here this case is not handled. Similarly, it is not possible to create a filter for both src port and dst port. For instance, filter on src port from [1-1000] and dst port = 80. The same kind of complexity is not possible for packet length, ICMP type, ICMP code.

There are some other known issues:

- The validation procedure depicted in **RFC 5575** is not available.

  This validation procedure has not been implemented, as this feature was not used in the existing setups you shared with us.
- The filtering action shaper value, if positive, is not used to apply shaping.

  If value is positive, the traffic is redirected to the wished destination, without any other action configured by Flowspec. It is recommended to configure Quality of Service if needed, more globally on a per interface basis.
- Upon an unexpected crash or other event, *zebra* may not have time to flush PBR contexts.

  That is to say `ipset`, `iptable` and `ip rule` contexts. This is also a consequence due to the fact that ip rule / ipset / iptables are not discovered at startup (not able to read appropriate contexts coming from Flowspec).

Appendix

More information with a public presentation that explains the design of Flowspec inside FRRouting.

[Presentation]

3.3.15 BGP fast-convergence support

Whenever BGP peer address becomes unreachable we must bring down the BGP session immediately. Currently only single-hop EBGP sessions are brought down immediately.IBGP and multi-hop EBGP sessions wait for hold-timer expiry to bring down the sessions.

This new configuration option helps user to teardown BGP sessions immediately whenever peer becomes unreachable.

`bgp fast-convergence`

This configuration is available at the bgp level. When enabled, configuration is applied to all the neighbors configured in that bgp instance.

```
router bgp 64496
 neighbor 10.0.0.2 remote-as 64496
 neighbor fd00::2 remote-as 64496
 bgp fast-convergence
!
address-family ipv4 unicast
 redistribute static
exit-address-family
!
address-family ipv6 unicast
 neighbor fd00::2 activate
exit-address-family
```

3.4 Babel

Babel is an interior gateway protocol that is suitable both for wired networks and for wireless mesh networks. Babel has been described as 'RIP on speed' – it is based on the same principles as RIP, but includes a number of refinements that make it react much faster to topology changes without ever counting to infinity, and allow it to perform reliable link quality estimation on wireless links. Babel is a double-stack routing protocol, meaning that a single Babel instance is able to perform routing for both IPv4 and IPv6.

FRR implements Babel as described in **RFC 6126**.

3.4.1 Configuring babeld

The *babeld* daemon can be invoked with any of the common options (*Common Invocation Options*).

The *zebra* daemon must be running before *babeld* is invoked. Also, if *zebra* is restarted then *babeld* must be too.

Configuration of *babeld* is done in its configuration file `babeld.conf`.

3.4.2 Babel configuration

`router babel`
Enable or disable Babel routing.

`babel diversity`
Enable or disable routing using radio frequency diversity. This is highly recommended in networks with many wireless nodes. If you enable this, you will probably want to set *babel diversity-factor* and *babel channel* below.

`babel diversity-factor (1-256)`
Sets the multiplicative factor used for diversity routing, in units of 1/256; lower values cause diversity to play a more important role in route selection. The default it 256, which means that diversity plays no role in route selection; you will probably want to set that to 128 or less on nodes with multiple independent radios.

`network IFNAME`
Enable or disable Babel on the given interface.

`babel <wired|wireless>`
Specifies whether this interface is wireless, which disables a number of optimisations that are only correct on wired interfaces. Specifying *wireless* (the default) is always correct, but may cause slower convergence and extra routing traffic.

`babel split-horizon`
: Specifies whether to perform split-horizon on the interface. Specifying `no babel split-horizon` is always correct, while `babel split-horizon` is an optimisation that should only be used on symmetric and transitive (wired) networks. The default is `babel split-horizon` on wired interfaces, and `no babel split-horizon` on wireless interfaces. This flag is reset when the wired/wireless status of an interface is changed.

`babel hello-interval (20-655340)`
: Specifies the time in milliseconds between two scheduled hellos. On wired links, Babel notices a link failure within two hello intervals; on wireless links, the link quality value is reestimated at every hello interval. The default is 4000 ms.

`babel update-interval (20-655340)`
: Specifies the time in milliseconds between two scheduled updates. Since Babel makes extensive use of triggered updates, this can be set to fairly high values on links with little packet loss. The default is 20000 ms.

`babel channel (1-254)`

`babel channel interfering`

`babel channel noninterfering`
: Set the channel number that diversity routing uses for this interface (see *babel diversity* above). Noninterfering interfaces are assumed to only interfere with themselves, interfering interfaces are assumed to interfere with all other channels except noninterfering channels, and interfaces with a channel number interfere with interfering interfaces and interfaces with the same channel number. The default is `babel channel interfering` for wireless interfaces, and `babel channel noninterfering` for wired interfaces. This is reset when the wired/wireless status of an interface is changed.

`babel rxcost (1-65534)`
: Specifies the base receive cost for this interface. For wireless interfaces, it specifies the multiplier used for computing the ETX reception cost (default 256); for wired interfaces, it specifies the cost that will be advertised to neighbours. This value is reset when the wired/wireless attribute of the interface is changed.

Note: Do not use this command unless you know what you are doing; in most networks, acting directly on the cost using route maps is a better technique.

`babel rtt-decay (1-256)`
: This specifies the decay factor for the exponential moving average of RTT samples, in units of 1/256. Higher values discard old samples faster. The default is 42.

`babel rtt-min (1-65535)`
: This specifies the minimum RTT, in milliseconds, starting from which we increase the cost to a neighbour. The additional cost is linear in (rtt - rtt-min). The default is 10 ms.

`babel rtt-max (1-65535)`
: This specifies the maximum RTT, in milliseconds, above which we don't increase the cost to a neighbour. The default is 120 ms.

`babel max-rtt-penalty (0-65535)`
: This specifies the maximum cost added to a neighbour because of RTT, i.e. when the RTT is higher or equal than rtt-max. The default is 150. Setting it to 0 effectively disables the use of a RTT-based cost.

`babel enable-timestamps`
: Enable or disable sending timestamps with each Hello and IHU message in order to compute RTT values. The default is *no babel enable-timestamps*.

`babel resend-delay (20-655340)`
: Specifies the time in milliseconds after which an 'important' request or update will be resent. The default is 2000 ms. You probably don't want to tweak this value.

`babel smoothing-half-life (0-65534)`

Specifies the time constant, in seconds, of the smoothing algorithm used for implementing hysteresis. Larger values reduce route oscillation at the cost of very slightly increasing convergence time. The value 0 disables hysteresis, and is suitable for wired networks. The default is 4 s.

3.4.3 Babel redistribution

`redistribute <ipv4|ipv6> KIND`

Specify which kind of routes should be redistributed into Babel.

3.4.4 Show Babel information

These commands dump various parts of *babeld*'s internal state.

`show babel route`

`show babel route A.B.C.D`

`show babel route X:X::X:X`

`show babel route A.B.C.D/M`

`show babel route X:X::X:X/M`

`show babel interface`

`show babel interface IFNAME`

`show babel neighbor`

`show babel parameters`

3.4.5 Babel debugging commands

simple: debug babel KIND simple: no debug babel KIND

`debug babel KIND`

Enable or disable debugging messages of a given kind. `KIND` can be one of:

- `common`
- `filter`
- `timeout`
- `interface`
- `route`
- `all`

Note: If you have compiled with the `NO_DEBUG` flag, then these commands aren't available.

3.4.6 Babel sample configuration file

```
debug babel common
!debug babel kernel
!debug babel filter
!debug babel timeout
!debug babel interface
!debug babel route
!debug babel all

router babel
! network wlan0
! network eth0
! redistribute ipv4 kernel
! no redistribute ipv6 static

! The defaults are fine for a wireless interface

!interface wlan0

! A few optimisation tweaks are optional but recommended on a wired interface
! Disable link quality estimation, enable split horizon processing, and
! increase the hello and update intervals.

!interface eth0
! babel wired
! babel split-horizon
! babel hello-interval 12000
! babel update-interval 36000

! log file /var/log/frr/babeld.log
log stdout
```

3.5 OpenFabric

OpenFabric, specified in *draft-white-openfabric-06.txt*, is a routing protocol derived from IS-IS, providing link-state routing with efficient flooding for topologies like spine-leaf networks.

FRR implements OpenFabric in a daemon called *fabricd*

3.5.1 Configuring fabricd

There are no *fabricd* specific options. Common options can be specified (*Common Invocation Options*) to *fabricd*. *fabricd* needs to acquire interface information from *zebra* in order to function. Therefore *zebra* must be running before invoking *fabricd*. Also, if *zebra* is restarted then *fabricd* must be too.

Like other daemons, *fabricd* configuration is done in an OpenFabric specific configuration file `fabricd.conf`.

3.5.2 OpenFabric router

To enable the OpenFabric routing protocol, an OpenFabric router needs to be created in the configuration:

`router openfabric WORD`
Enable or disable the OpenFabric process by specifying the OpenFabric domain with 'WORD'.

`net XX.XXXX. ... .XXX.XX`
Set/Unset network entity title (NET) provided in ISO format.

`domain-password [clear | md5] <password>`
Configure the authentication password for a domain, as clear text or md5 one.

`attached-bit [receive ignore | send]`
Set attached bit for inter-area traffic:

- receive If LSP received with attached bit set, create default route to neighbor
- send If L1|L2 router, set attached bit in LSP sent to L1 router

`log-adjacency-changes`
Log changes in adjacency state.

`set-overload-bit`
Set overload bit to avoid any transit traffic.

`purge-originator`
Enable or disable **RFC 6232** purge originator identification.

`fabric-tier (0-14)`
Configure a static tier number to advertise as location in the fabric

3.5.3 OpenFabric Timer

`lsp-gen-interval (1-120)`
Set minimum interval in seconds between regenerating same LSP.

`lsp-refresh-interval (1-65235)`
Set LSP refresh interval in seconds.

`max-lsp-lifetime (360-65535)`
Set LSP maximum LSP lifetime in seconds.

`spf-interval (1-120)`
Set minimum interval between consecutive SPF calculations in seconds.

3.5.4 OpenFabric interface

`ip router openfabric WORD`

Activate OpenFabric on this interface. Note that the name of OpenFabric instance must be the same as the one used to configure the routing process (see command *`router openfabric WORD`*).

`openfabric csnp-interval (1-600)`
Set CSNP interval in seconds.

`openfabric hello-interval (1-600)`
Set Hello interval in seconds.

openfabric hello-multiplier (2-100)
Set multiplier for Hello holding time.

openfabric metric (0-16777215)
Set interface metric value.

openfabric passive
Configure the passive mode for this interface.

openfabric password [clear | md5] <password>
Configure the authentication password (clear or encoded text) for the interface.

openfabric psnp-interval (1-120)
Set PSNP interval in seconds.

3.5.5 Showing OpenFabric information

show openfabric summary
Show summary information about OpenFabric.

show openfabric hostname
Show which hostnames are associated with which OpenFabric system ids.

show openfabric interface

show openfabric interface detail

show openfabric interface <interface name>
Show state and configuration of specified OpenFabric interface, or all interfaces if no interface is given with or without details.

show openfabric neighbor

show openfabric neighbor <System Id>

show openfabric neighbor detail
Show state and information of specified OpenFabric neighbor, or all neighbors if no system id is given with or without details.

show openfabric database

show openfabric database [detail]

show openfabric database <LSP id> [detail]

show openfabric database detail <LSP id>
Show the OpenFabric database globally, for a specific LSP id without or with details.

show openfabric topology
Show calculated OpenFabric paths and associated topology information.

3.5.6 Debugging OpenFabric

`debug openfabric adj-packets`
OpenFabric Adjacency related packets.

`debug openfabric checksum-errors`
OpenFabric LSP checksum errors.

`debug openfabric events`
OpenFabric Events.

`debug openfabric local-updates`
OpenFabric local update packets.

`debug openfabric lsp-gen`
Generation of own LSPs.

`debug openfabric lsp-sched`
Debug scheduling of generation of own LSPs.

`debug openfabric packet-dump`
OpenFabric packet dump.

`debug openfabric protocol-errors`
OpenFabric LSP protocol errors.

`debug openfabric route-events`
OpenFabric Route related events.

`debug openfabric snp-packets`
OpenFabric CSNP/PSNP packets.

`debug openfabric spf-events`

`debug openfabric spf-statistics`

`debug openfabric spf-triggers`
OpenFabric Shortest Path First Events, Timing and Statistic Data and triggering events.

`debug openfabric update-packets`
Update-related packets.

`show debugging openfabric`
Print which OpenFabric debug levels are active.

3.5.7 Sample configuration

A simple example:

```
!
interface lo
 ip address 192.0.2.1/32
 ip router openfabric 1
 ipv6 address 2001:db8::1/128
 ipv6 router openfabric 1
!
interface eth0
 ip router openfabric 1
 ipv6 router openfabric 1
```

(continues on next page)

(continued from previous page)

```
!
interface eth1
 ip router openfabric 1
 ipv6 router openfabric 1
!
router openfabric 1
 net 49.0000.0000.0001.00
```

Alternative example:

```
hostname fabricd

router openfabric DEAD
  net 47.0023.0000.0003.0300.0100.0102.0304.0506.00
  lsp-lifetime 65535

  hostname isisd-router
  domain-password foobar

interface eth0
 ip router openfabric DEAD
 openfabric hello-interval 5
 openfabric lsp-interval 1000

! -- optional
openfabric retransmit-interval 10
openfabric retransmit-throttle-interval
```

3.6 LDP

The *ldpd* daemon is a standardised protocol that permits exchanging MPLS label information between MPLS devices. The LDP protocol creates peering between devices, so as to exchange that label information. This information is stored in MPLS table of *zebra*, and it injects that MPLS information in the underlying system (Linux kernel or OpenBSD system for instance). *ldpd* provides necessary options to create a Layer 2 VPN across MPLS network. For instance, it is possible to interconnect several sites that share the same broadcast domain.

FRR implements LDP as described in **RFC 5036**; other LDP standard are the following ones: **RFC 6720**, **RFC 6667**, **RFC 5919**, **RFC 5561**, **RFC 7552**, **RFC 4447**. Because MPLS is already available, FRR also supports **RFC 3031**.

3.6.1 Running Ldpd

The *ldpd* daemon can be invoked with any of the common options (*Common Invocation Options*).

--ctl_socket

This option allows you to override the path to the ldpd.sock file used to control this daemon. If specified this option overrides the -N option path addition.

The *zebra* daemon must be running before *ldpd* is invoked.

Configuration of *ldpd* is done in its configuration file `ldpd.conf`.

3.6.2 Understanding LDP principles

Let's first introduce some definitions that permit understand better the LDP protocol:

- *LSR* : Labeled Switch Router. Networking devices handling labels used to forward traffic between and through them.
- ***LER*** [Labeled Edge Router. A Labeled edge router is located at the edge of] an MPLS network, generally between an IP network and an MPLS network.

LDP aims at sharing label information across devices. It tries to establish peering with remote LDP capable devices, first by discovering using UDP port 646 , then by peering using TCP port 646. Once the TCP session is established, the label information is shared, through label advertisements.

There are different methods to send label advertisement modes. The implementation actually supports the following : Liberal Label Retention + Downstream Unsolicited + Independent Control. The other advertising modes are depicted below, and compared with the current implementation.

- Liberal label retention versus conservative mode In liberal mode, every label sent by every LSR is stored in the MPLS table. In conservative mode, only the label that was sent by the best next hop (determined by the IGP metric) for that particular FEC is stored in the MPLS table.
- Independent LSP Control versus ordered LSP Control MPLS has two ways of binding labels to FEC's; either through ordered LSP control, or independent LSP control. Ordered LSP control only binds a label to a FEC if it is the egress LSR, or the router received a label binding for a FEC from the next hop router. In this mode, an MPLS router will create a label binding for each FEC and distribute it to its neighbors so long as he has a entry in the RIB for the destination. In the other mode, label bindings are made without any dependencies on another router advertising a label for a particular FEC. Each router makes it own independent decision to create a label for each FEC. By default IOS uses Independent LSP Control, while Juniper implements the Ordered Control. Both modes are interoperable, the difference is that Ordered Control prevent blackholing during the LDP convergence process, at cost of slowing down the convergence itself
- unsolicited downstream versus downstream on demand Downstream on demand label distribution is where an LSR must explicitly request that a label be sent from its downstream router for a particular FEC. Unsolicited label distribution is where a label is sent from the downstream router without the original router requesting it.

3.6.3 LDP Configuration

`mpls ldp`
: Enable or disable LDP daemon

`router-id A.B.C.D`
: The following command located under MPLS router node configures the MPLS router-id of the local device.

`ordered-control`
: Configure LDP Ordered Label Distribution Control.

`address-family [ipv4 | ipv6]`
: Configure LDP for IPv4 or IPv6 address-family. Located under MPLS route node, this subnode permits configuring the LDP neighbors.

`interface IFACE`
: Located under MPLS address-family node, use this command to enable or disable LDP discovery per interface. IFACE stands for the interface name where LDP is enabled. By default it is disabled. Once this command executed, the address-family interface node is configured.

`discovery transport-address A.B.C.D | A:B::C:D`
: Located under mpls address-family interface node, use this command to set the IPv4 or IPv6 transport-address used by the LDP protocol to talk on this interface.

ttl-security disable
Located under the LDP address-family node, use this command to disable the GTSM procedures described in RFC 6720 (for the IPv4 address-family) and RFC 7552 (for the IPv6 address-family).

Since GTSM is mandatory for LDPv6, the only effect of disabling GTSM for the IPv6 address-family is that *ldpd* will not discard packets with a hop limit below 255. This may be necessary to interoperate with older implementations. Outgoing packets will still be sent using a hop limit of 255 for maximum compatibility.

If GTSM is enabled, multi-hop neighbors should have either GTSM disabled individually or configured with an appropriate ttl-security hops distance.

neighbor A.B.C.D password PASSWORD
The following command located under MPLS router node configures the router of a LDP device. This device, if found, will have to comply with the configured password. PASSWORD is a clear text password wit its digest sent through the network.

neighbor A.B.C.D holdtime HOLDTIME
The following command located under MPLS router node configures the holdtime value in seconds of the LDP neighbor ID. Configuring it triggers a keepalive mechanism. That value can be configured between 15 and 65535 seconds. After this time of non response, the LDP established session will be considered as set to down. By default, no holdtime is configured for the LDP devices.

neighbor A.B.C.D ttl-security disable
Located under the MPLS LDP node, use this command to override the global configuration and enable/disable GTSM for the specified neighbor.

neighbor A.B.C.D ttl-security hops (1-254)
Located under the MPLS LDP node, use this command to set the maximum number of hops the specified neighbor may be away. When GTSM is enabled for this neighbor, incoming packets are required to have a TTL/hop limit of 256 minus this value, ensuring they have not passed through more than the expected number of hops. The default value is 1.

discovery hello holdtime HOLDTIME

discovery hello interval INTERVAL
INTERVAL value ranges from 1 to 65535 seconds. Default value is 5 seconds. This is the value between each hello timer message sent. HOLDTIME value ranges from 1 to 65535 seconds. Default value is 15 seconds. That value is added as a TLV in the LDP messages.

dual-stack transport-connection prefer ipv4
When *ldpd* is configured for dual-stack operation, the transport connection preference is IPv6 by default (as specified by **RFC 7552**). On such circumstances, *ldpd* will refuse to establish TCP connections over IPv4. You can use above command to change the transport connection preference to IPv4. In this case, it will be possible to distribute label mappings for IPv6 FECs over TCPv4 connections.

3.6.4 Show LDP Information

These commands dump various parts of *ldpd*.

show mpls ldp neighbor [A.B.C.D]
This command dumps the various neighbors discovered. Below example shows that local machine has an operation neighbor with ID set to 1.1.1.1.

```
west-vm# show mpls ldp neighbor
AF   ID              State       Remote Address    Uptime
ipv4 1.1.1.1         OPERATIONAL 1.1.1.1          00:01:37
west-vm#
```

show mpls ldp neighbor [A.B.C.D] capabilities

show mpls ldp neighbor [A.B.C.D] detail

Above commands dump other neighbor information.

show mpls ldp discovery [detail]

show mpls ldp ipv4 discovery [detail]

show mpls ldp ipv6 discovery [detail]

Above commands dump discovery information.

show mpls ldp ipv4 interface

show mpls ldp ipv6 interface

Above command dumps the IPv4 or IPv6 interface per where LDP is enabled. Below output illustrates what is dumped for IPv4.

```
west-vm# show mpls ldp ipv4 interface
AF   Interface    State  Uptime   Hello Timers  ac
ipv4 eth1        ACTIVE 00:08:35 5/15          0
ipv4 eth3        ACTIVE 00:08:35 5/15          1
```

show mpls ldp ipv4|ipv6 binding

Above command dumps the binding obtained through MPLS exchanges with LDP.

```
west-vm# show mpls ldp ipv4 binding
AF   Destination          Nexthop         Local Label Remote Label  In Use
ipv4 1.1.1.1/32           1.1.1.1         16          imp-null         yes
ipv4 2.2.2.2/32           1.1.1.1         imp-null    16                no
ipv4 10.0.2.0/24          1.1.1.1         imp-null    imp-null          no
ipv4 10.115.0.0/24        1.1.1.1         imp-null    17                no
ipv4 10.135.0.0/24        1.1.1.1         imp-null    imp-null          no
ipv4 10.200.0.0/24        1.1.1.1         17          imp-null         yes
west-vm#
```

3.6.5 LDP debugging commands

debug mpls ldp KIND

Enable or disable debugging messages of a given kind. `KIND` can be one of:

- `discovery`
- `errors`
- `event`
- `labels`
- `messages`
- `zebra`

3.6.6 Sample configuration

Below configuration gives a typical MPLS configuration of a device located in a MPLS backbone. LDP is enabled on two interfaces and will attempt to peer with two neighbors with router-id set to either 1.1.1.1 or 3.3.3.3.

```
mpls ldp
 router-id 2.2.2.2
 neighbor 1.1.1.1 password test
 neighbor 3.3.3.3 password test
 !
 address-family ipv4
  discovery transport-address 2.2.2.2
  !
  interface eth1
  !
  interface eth3
  !
 exit-address-family
 !
```

Deploying LDP across a backbone generally is done in a full mesh configuration topology. LDP is typically deployed with an IGP like OSPF, that helps discover the remote IPs. Below example is an OSPF configuration extract that goes with LDP configuration

```
router ospf
 ospf router-id 2.2.2.2
  network 0.0.0.0/0 area 0
 !
```

Below output shows the routing entry on the LER side. The OSPF routing entry (10.200.0.0) is associated with Label entry (17), and shows that MPLS push action that traffic to that destination will be applied.

```
north-vm# show ip route
Codes: K - kernel route, C - connected, S - static, R - RIP,
       O - OSPF, I - IS-IS, B - BGP, E - EIGRP, N - NHRP,
       T - Table, v - VNC, V - VNC-Direct, A - Babel, D - SHARP,
       F - PBR,
       > - selected route, * - FIB route

O>* 1.1.1.1/32 [110/120] via 10.115.0.1, eth2, label 16, 00:00:15
O>* 2.2.2.2/32 [110/20] via 10.115.0.1, eth2, label implicit-null, 00:00:15
O   3.3.3.3/32 [110/10] via 0.0.0.0, loopback1 onlink, 00:01:19
C>* 3.3.3.3/32 is directly connected, loopback1, 00:01:29
O>* 10.0.2.0/24 [110/11] via 10.115.0.1, eth2, label implicit-null, 00:00:15
O   10.100.0.0/24 [110/10] is directly connected, eth1, 00:00:32
C>* 10.100.0.0/24 is directly connected, eth1, 00:00:32
O   10.115.0.0/24 [110/10] is directly connected, eth2, 00:00:25
C>* 10.115.0.0/24 is directly connected, eth2, 00:00:32
O>* 10.135.0.0/24 [110/110] via 10.115.0.1, eth2, label implicit-null, 00:00:15
O>* 10.200.0.0/24 [110/210] via 10.115.0.1, eth2, label 17, 00:00:15
north-vm#
```

Additional example demonstrating use of some miscellaneous config options:

```
interface eth0
!
interface eth1
!
interface lo
!
mpls ldp
 dual-stack cisco-interop
 neighbor 10.0.1.5 password opensourcerouting
 neighbor 172.16.0.1 password opensourcerouting
 !
 address-family ipv4
  discovery transport-address 10.0.1.1
  label local advertise explicit-null
  !
  interface eth0
  !
  interface eth1
  !
 !
 address-family ipv6
  discovery transport-address 2001:db8::1
  !
  interface eth1
  !
 !
!
l2vpn ENG type vpls
 bridge br0
 member interface eth2
 !
 member pseudowire mpw0
  neighbor lsr-id 1.1.1.1
  pw-id 100
 !
!
```

3.7 EIGRP

DUAL The *Diffusing Update ALgorithm*, a *Bellman-Ford* based routing algorithm used by EIGRP.

EIGRP – Routing Information Protocol is widely deployed interior gateway routing protocol. EIGRP was developed in the 1990's. EIGRP is a *distance-vector* protocol and is based on the *DUAL* algorithms. As a distance-vector protocol, the EIGRP router send updates to its neighbors as networks change, thus allowing the convergence to a known topology.

eigrpd supports EIGRP as described in RFC7868

3.7.1 Starting and Stopping eigrpd

The default configuration file name of *eigrpd*'s is `eigrpd.conf`. When invocation *eigrpd* searches directory /etc/frr. If `eigrpd.conf` is not there next search current directory. If an integrated config is specified configuration is written into `frr.conf`.

The EIGRP protocol requires interface information maintained by *zebra* daemon. So running *zebra* is mandatory to run *eigrpd*. Thus minimum sequence for running EIGRP is:

```
# zebra -d
# eigrpd -d
```

Please note that *zebra* must be invoked before *eigrpd*.

To stop *eigrpd*, please use:

```
kill `cat /var/run/frr/eigrpd.pid`
```

Certain signals have special meanings to *eigrpd*.

Signal	Meaning
SIGHUP & SIGUSR1	Rotate the log file
SIGINT & SIGTERM	Sweep all installed EIGRP routes and gracefully terminate

eigrpd invocation options. Common options that can be specified (*Common Invocation Options*).

3.7.2 EIGRP Configuration

`router eigrp (1-65535) [vrf NAME]`

The *router eigrp* command is necessary to enable EIGRP. To disable EIGRP, use the *no router eigrp (1-65535)* command. EIGRP must be enabled before carrying out any of the EIGRP commands. Specify vrf NAME if you want eigrp to work within the specified vrf.

`network NETWORK`

Set the EIGRP enable interface by *network*. The interfaces which have addresses matching with *network* are enabled.

This group of commands either enables or disables EIGRP interfaces between certain numbers of a specified network address. For example, if the network for 10.0.0.0/24 is EIGRP enabled, this would result in all the addresses from 10.0.0.0 to 10.0.0.255 being enabled for EIGRP. The *no network* command will disable EIGRP for the specified network.

Below is very simple EIGRP configuration. Interface *eth0* and interface which address match to *10.0.0.0/8* are EIGRP enabled.

```
!
router eigrp 1
 network 10.0.0.0/8
!
```

`passive-interface (IFNAME|default)`

This command sets the specified interface to passive mode. On passive mode interface, all receiving packets are ignored and eigrpd does not send either multicast or unicast EIGRP packets except to EIGRP neighbors specified with *neighbor* command. The interface may be specified as *default* to make eigrpd default to passive on all interfaces.

The default is to be passive on all interfaces.

3.7.3 How to Announce EIGRP route

Redistribute routes into EIGRP:

`redistribute <babel|bgp|connected|isis|kernel|openfabric|ospf|rip|sharp|static|table> [metric (1-429496`

The `redistribute` family of commands imports routing information from other sources into EIGRP's tables. Redistribution may be disabled with the `no` form of the commands.

Note that connected routes on interfaces EIGRP is enabled on are announced by default.

Optionally, various EIGRP metrics may be specified. These metrics will be applied to the imported routes.

3.7.4 Show EIGRP Information

`show ip eigrp [vrf NAME] topology`

Display current EIGRP status.

```
eigrpd> **show ip eigrp topology**
# show ip eigrp topo

EIGRP Topology Table for AS(4)/ID(0.0.0.0)

Codes: P - Passive, A - Active, U - Update, Q - Query, R - Reply
       r - reply Status, s - sia Status

P  10.0.2.0/24, 1 successors, FD is 256256, serno: 0
       via Connected, enp0s3
```

`show ip eigrp [vrf NAME] interface`

Display the list of interfaces associated with a particular eigrp instance.

`show ip eigrp [vrf NAME] neighbor`

Display the list of neighbors that have been established within a particular eigrp instance.

3.7.5 EIGRP Debug Commands

Debug for EIGRP protocol.

`debug eigrp packets`

Debug eigrp packets

`debug eigrp` will show EIGRP packets that are sent and received.

`debug eigrp transmit`

Debug eigrp transmit events

`debug eigrp transmit` will display detailed information about the EIGRP transmit events.

`show debugging eigrp`

Display *eigrpd*'s debugging option.

`show debugging eigrp` will show all information currently set for eigrpd debug.

3.7.6 Sample configuration

```
hostname eigrpd
password zebra
enable password please-set-at-here
!
router eigrp 4453
  network 192.168.1.0/24
!
log stdout
```

3.8 EVPN

EVPN stands for Ethernet Virtual Private Network. This is an extension of BGP that enables the signaling of bridged (L2) and routed (L3) VPNs over a common network. EVPN is described in **RFC 7432** and is updated by several additional RFCs and IETF drafts including **RFC 9135** (Integrated Routing and Bridging in Ethernet VPN), **RFC 9136** (IP Prefix Advertisement in Ethernet VPN), **RFC 8584** (Framework for Ethernet VPN Designated Forwarder Election Extensibility), and **RFC 8365** (A Network Virtualization Overlay Solution Using Ethernet VPN). FRR supports All-Active Layer-2 Multihoming for devices (MHD) via LACP Ethernet Segments as well as both Symmetric and Asymmetric IRB. FRR implements MAC-VRFs using a "VLAN-Based Service Interface" (**RFC 7432**) and performs processing of Symmetric IRB routes following the "Interface-less IP-VRF-to-IP-VRF Model" (**RFC 9136**).

3.8.1 EVPN Concepts

BGP-EVPN is the control plane for the transport of Ethernet frames, regardless of whether those frames are bridged or routed. In the case of a VLAN-Based Service Interface with VXLAN encap, a single VNI is used to represent an EVPN Instance (EVI) and will have its own Route Distinguisher and set of Import/Export Route-Targets.

A VNI is considered to be either Layer-2 (tied to a MAC-VRF) or Layer-3 (tied to an IP-VRF), which indicates what kind of information is represented by the VRF. An IP-VRF represents a routing table (operating in much the same way as a VRF traditionally operates in L3VPN), while a MAC-VRF represents a bridging table i.e. MAC (fdb) and ARP/NDP entries.

A MAC-VRF can be thought of as a VLAN with or without an SVI associated with it. An SVI is a Layer-3 interface bound to a bridging domain. In Linux an SVI can either be a traditional bridge or a VLAN subinterface of a VLAN-aware bridge. If there is an SVI for the VLAN, ARP/NDP entries can be bound to the MACs within the broadcast domain. Without an SVI, the VLAN operates in traditional L2 fashion and MACs are the only type of host addresses known within the VLAN.

In the same way that there can be a many-to-one relationship of SVIs to a VRF, there can also be a many-to-one relationship of MAC-VRFs (L2VNIs) to an IP-VRF (L3VNI). In FRR the L3VNI association for an L2VNI is determined by the presence of an SVI for the VLAN and the VRF membership of the SVI. If an L2VNI does not have an SVI or its SVI is not enslaved to a VRF, the L2VNI will be associated with the "default" VRF. If an L2VNI has an SVI whose master device is a VRF, then that L2VNI will be associated with its master VRF.

3.8.2 FRR Configuration

FRR learns about the system's Linux network interface configuration from the kernel via Netlink, however it does not manage network interfaces directly. The following sections will include examples of Linux interface configurations that are compatible with FRR's EVPN implementation. While there are multiple interface managers that can setup a proper kernel config (e.g. ifupdown2), these examples will use iproute2 to add/configure the interfaces.

All of the examples will follow the same basic setup but use different, yet compatible, interface configurations.

In this example we will setup the following:

- An IP-VRF named vrf1, associated with L3VNI 100
- An IP-VRF named vrf2, associated with L3VNI 200
- An IP-VRF named vrf3, with no L3VNI associations
- A MAC-VRF using VLAN 10, associated with L2VNI 110 and IP-VRF vrf1
- A MAC-VRF using VLAN 20, associated with L2VNI 220 and IP-VRF vrf2
- A MAC-VRF using VLAN 30, associated with L2VNI 330 and IP-VRF vrf3
- A MAC-VRF using VLAN 40, associated with L2VNI 440 and IP-VRF default
- A MAC-VRF using VLAN 50, associated with L2VNI 550 and operating L2-Only

Sample Configuration

This is a sample FRR configuration that implements the above EVPN environment. The first snippet will be the config in its entirety, then each config element will be explained individually later in the document.

The following snippet will result in a functional EVPN control plane if the corresponding Linux interface configuration is correct, compatible, and active:

```
vrf vrf1
 vni 100
exit-vrf
!
vrf vrf2
 vni 200
exit-vrf
!
router bgp 4200000000
 neighbor 192.168.122.12 remote-as internal
 !
 address-family ipv4 unicast
  network 100.64.0.1/32
 exit-address-family
 !
 address-family l2vpn evpn
  neighbor 192.168.122.12 activate
  advertise-all-vni
  advertise-svi-ip
 exit-address-family
exit
!
router bgp 4200000000 vrf vrf1
```

(continues on next page)

(continued from previous page)

```
 !
 address-family ipv4 unicast
  redistribute static
 exit-address-family
 !
 address-family ipv6 unicast
  redistribute static
 exit-address-family
 !
 address-family l2vpn evpn
  advertise ipv4 unicast
  advertise ipv6 unicast
 exit-address-family
exit
!
router bgp 4200000000 vrf vrf2
 !
 address-family ipv4 unicast
  redistribute static
 exit-address-family
 !
 address-family ipv6 unicast
  redistribute static
 exit-address-family
 !
 address-family l2vpn evpn
  advertise ipv4 unicast
  advertise ipv6 unicast
 exit-address-family
exit
```

A VRF will get its L3VNI association as a result of the `vni` command under the `vrf` stanza. Until this L3VNI association is made, zebra will discover the VNI from netlink but will consider it to be an L2VNI. The current L2 vs L3 context of a VNI can be seen in the output of `show evpn vni`.

In this configuration we are telling zebra to consider VXLAN-ID 100 to be the L3VNI for vrf1 and VXLAN-ID 200 to be the L3VNI for vrf2.

```
vrf vrf1
 vni 100
exit-vrf
!
vrf vrf2
 vni 200
exit-vrf
```

The VTEP-IP (100.64.0.1) needs to be reachable by other VTEPs in the EVPN environment in order for VXLAN decapsulation to function. In this example we will advertise our local VTEP-IP using BGP (via the `network` statement), but static routes or other routing protocols like IS-IS or OSPF can also be used.

In order to enable EVPN for a BGP instance, we must use the command `advertise-all-vni`. In this example we will be using the default VRF to carry the l2vpn evpn address-family, so we will enable EVPN for the default VRF.

In this example, we plan to exchange EVPN routes with 192.168.122.12, so we will activate the l2vpn evpn address-family for this peer in order to allow EVPN NLRI to be advertised and received.

The `advertise-svi-ip` command also belongs in the BGP instance where EVPN is enabled. This command tells FRR to originate "self" Type-2 routes for all the MAC/IP pairs associated with the local SVI interfaces.

```
router bgp 4200000000
 neighbor 192.168.122.12 remote-as internal
 !
 address-family ipv4 unicast
  network 100.64.0.1/32
 exit-address-family
 !
 address-family l2vpn evpn
  neighbor 192.168.122.12 activate
  advertise-all-vni
  advertise-svi-ip
 exit-address-family
exit
```

IPv4 and IPv6 BGP Prefixes from an IP-VRF are not exported to EVPN as Type-5 routes until the respective `advertise <afi> unicast` command has been configured in the BGP instance of the VRF in question. All routes in the BGP RIB (locally originated, learned from a peer, or leaked from another VRF) will be eligible to be exported to EVPN so long as they are valid and selected in the VRF's unicast table.

In this example, the BGP instances for vrf1 and vrf2 will have their static routes redistributed into the BGP loc-rib for the ipv4 unicast and ipv6 unicast address-families via the `redistribute static` statements. These unicast prefixes will then be exported into EVPN as Type-5 routes as a result of the `advertise ipv4 unicast` and `advertise ipv6 unicast` commands.

```
router bgp 4200000000 vrf vrf1
 !
 address-family ipv4 unicast
  redistribute static
 exit-address-family
 !
 address-family ipv6 unicast
  redistribute static
 exit-address-family
 !
 address-family l2vpn evpn
  advertise ipv4 unicast
  advertise ipv6 unicast
 exit-address-family
exit
!
router bgp 4200000000 vrf vrf2
 !
 address-family ipv4 unicast
  redistribute static
 exit-address-family
 !
 address-family ipv6 unicast
  redistribute static
 exit-address-family
 !
```

(continues on next page)

(continued from previous page)

```
 address-family l2vpn evpn
  advertise ipv4 unicast
  advertise ipv6 unicast
 exit-address-family
exit
```

3.8.3 Linux Interface Configuration

The Linux kernel offers several options for configuring netdevices for an EVPN-VXLAN environment. The following section will include samples of a few netdev configurations that are compatible with FRR which implement the environment described above.

Some high-level config considerations:

- The local VTEP-IP should always be set to a reachable IP on the lo device.
- An L3VNI should always have an SVI (aka the L3-SVI).
- An L3-SVI should not be assigned an IP address, link-local or otherwise.
  - IPv6 address autoconfiguration can be disabled via `addrgenmode none`.
- An SVI for an L2VNI is only needed for routing (IRB) or ARP/ND suppression.
  - ARP/ND suppression is a kernel function, it is not managed by FRR.
  - ARP/ND suppression is enabled per bridge_slave via `neigh_suppress`.
  - ARP/ND suppression should only be enabled on vxlan interfaces.
  - IPv4/IPv6 forwarding should be disabled on SVIs not used for routing (IRB).
- Dynamic MAC/VTEP learning should be disabled on VXLAN interfaces used in EVPN.
  - Dynamic MAC learning is a function of the kernel bridge driver, not FRR.
  - Dynamic MAC learning is toggled per bridge_slave via `learning {on|off}`.
  - Dynamic VTEP learning is a function of the kernel vxlan driver, not FRR.
  - Dynamic VTEP learning is toggled per vxlan interface via `[no]learning`.
- The VXLAN interfaces should not have a `remote` VTEP defined.
  - Remote VTEPs are learned via EVPN, so static VTEPs are unnecessary.

Traditional Bridges and Traditional VXLAN Devices

In the traditional bridge model, we use a separate `bridge` interface per MAC-VRF which acts as the SVI for that broadcast domain. A bridge is considered "traditional" if `vlan_filtering` is set to `0` (disabled) which indicates the bridge only has one broadcast domain which does not consider VLAN tags. Similarly, only one VNI is carried by each "traditional" `vxlan` interface. So in this deployment model, each VXLAN-enabled broadcast domain will have one traditional vxlan interface enslaved to one traditional bridge.

Bridges created for an L3VNI broadcast domain should only have one member: the L3VNI vxlan device. Bridges created for an L2VNI broadcast domain generally have multiple members: the L2VNI vxlan device, plus any host/network ports where the L2 domain will be carried.

To carry the broadcast domains of multiple traditional bridges over the same host/network port, a tagged `vlan` sub-interface of the port must be created per broadcast domain. The vlan sub-interfaces would then be enslaved to the traditional bridge, ensuring that only packets tagged with the expected VID are associated with the expected broadcast domain.

```
###################
## vxlan vtep-ip ##
###################
ip addr add 100.64.0.1/32 dev lo

#############################
## ip-vrf vrf1 / l3vni 100 ##
#############################
ip link add vrf1 type vrf table 1100
ip link set vrf1 up
ip link add br100 type bridge
ip link set br100 master vrf1 addrgenmode none
ip link set br100 addr aa:bb:cc:00:00:64
ip link add vni100 type vxlan local 100.64.0.1 dstport 4789 id 100 nolearning
ip link set vni100 master br100 addrgenmode none
ip link set vni100 type bridge_slave neigh_suppress on learning off
ip link set vni100 up
ip link set br100 up

#############################
## ip-vrf vrf2 / l3vni 200 ##
#############################
ip link add vrf2 type vrf table 1200
ip link set vrf2 up
ip link add br200 type bridge
ip link set br200 master vrf2 addrgenmode none
ip link set br200 addr aa:bb:cc:00:00:c8
ip link add vni200 type vxlan local 100.64.0.1 dstport 4789 id 200 nolearning
ip link set vni200 master br200 addrgenmode none
ip link set vni200 type bridge_slave neigh_suppress on learning off
ip link set vni200 up
ip link set br200 up

#################
## ip-vrf vrf3 ##
#################
ip link add vrf3 type vrf table 1300
ip link set vrf3 up

###############
## l2vni 110 ##
###############
ip link add br10 type bridge
ip link set br10 master vrf1
ip link set br10 addr aa:bb:cc:00:00:6e
ip addr add 10.0.10.1/24 dev br10
ip addr add 2001:db8:0:10::1/64 dev br10
ip link add vni110 type vxlan local 100.64.0.1 dstport 4789 id 110 nolearning
ip link set vni110 master br10 addrgenmode none
ip link set vni110 type bridge_slave neigh_suppress on learning off
```

(continues on next page)

(continued from previous page)

```
ip link set vni110 up
ip link set br10 up

###############
## l2vni 220 ##
###############
ip link add br20 type bridge
ip link set br20 master vrf2
ip link set br20 addr aa:bb:cc:00:00:dc
ip addr add 10.0.20.1/24 dev br20
ip addr add 2001:db8:0:20::1/64 dev br20
ip link add vni220 type vxlan local 100.64.0.1 dstport 4789 id 220 nolearning
ip link set vni220 master br20 addrgenmode none
ip link set vni220 type bridge_slave neigh_suppress on learning off
ip link set vni220 up
ip link set br20 up

###############
## l2vni 330 ##
###############
ip link add br30 type bridge
ip link set br30 master vrf3
ip link set br30 addr aa:bb:cc:00:01:4a
ip addr add 10.0.30.1/24 dev br30
ip addr add 2001:db8:0:30::1/64 dev br30
ip link add vni330 type vxlan local 100.64.0.1 dstport 4789 id 330 nolearning
ip link set vni330 master br30 addrgenmode none
ip link set vni330 type bridge_slave neigh_suppress on learning off
ip link set vni330 up
ip link set br30 up

###############
## l2vni 440 ##
###############
ip link add br40 type bridge
ip link set br40 addr aa:bb:cc:00:01:b8
ip addr add 10.0.40.1/24 dev br40
ip addr add 2001:db8:0:40::1/64 dev br40
ip link add vni440 type vxlan local 100.64.0.1 dstport 4789 id 440 nolearning
ip link set vni440 master br40 addrgenmode none
ip link set vni440 type bridge_slave neigh_suppress on learning off
ip link set vni440 up
ip link set br40 up

###############
## l2vni 550 ##
###############
ip link add br50 type bridge
ip link set br50 addrgenmode none
ip link set br50 addr aa:bb:cc:00:02:26
ip link add vni550 type vxlan local 100.64.0.1 dstport 4789 id 550 nolearning
ip link set vni550 master br50 addrgenmode none
```

(continues on next page)

(continued from previous page)

```
ip link set vni550 type bridge_slave neigh_suppress on learning off
sysctl -w net.ipv4.conf.br50.forwarding=0
sysctl -w net.ipv6.conf.br50.forwarding=0
ip link set vni550 up
ip link set br50 up

##################
## create vlan subinterface of eth0 for each l2vni vlan and enslave each
## subinterface to the corresponding bridge
##################
ip link set eth0 up
for i in 10 20 30 40 50; do
   ip link add link eth0 name eth0.$i type vlan id $i;
   ip link set eth0.$i master br$i;
   ip link set eth0.$i up;
done
```

To begin with, it creates a `vrf` interface named "vrf1" that is bound to the kernel routing table with ID 1100. This will represent the IP-VRF "vrf1" which we will later allocate an L3VNI for.

```
ip link add vrf1 type vrf table 1100
```

This block creates a traditional `bridge` interface named "br100", binds it to the VRF named "vrf1", disables IPv6 address autoconfiguration, and statically defines the MAC address of "br100". This traditional bridge is used for the L3VNI broadcast domain mapping to VRF "vrf1", i.e. "br100" is vrf1's L3-SVI.

```
ip link add br100 type bridge
ip link set br100 master vrf1 addrgenmode none
ip link set br100 addr aa:bb:cc:00:00:64
```

Here a traditional `vxlan` interface is created with the name "vni100" which uses a VTEP-IP of 100.64.0.1, carries VNI 100, and has Dynamic VTEP learning disabled. IPv6 address autoconfiguration is disabled for "vni100", then the interface is enslaved to "br100", ARP/ND suppression is enabled, and Dynamic MAC Learning is disabled.

```
ip link add vni100 type vxlan local 100.64.0.1 dstport 4789 id 100 nolearning
ip link set vni100 master br100 addrgenmode none
ip link set vni100 type bridge_slave neigh_suppress on learning off
```

This completes the necessary configuration for a VRF and L3VNI.

Here a traditional bridge named "br10" is created. We add "br10" to "vrf1" by setting "vrf1" as the `master` of "br10". It is not necessary to set the SVI MAC statically, but it is done here for consistency's sake. Since "br10" will be used for routing, IPv4 and IPv6 addresses are also added to the SVI.

```
ip link add br10 type bridge
ip link set br10 master vrf1
ip link set br10 addr aa:bb:cc:00:00:6e
ip addr add 10.0.10.1/24 dev br10
ip addr add 2001:db8:0:10::1/64 dev br10
```

If the SVI will not be used for routing, IP addresses should not be assigned to the SVI interface and IPv4/IPv6 "forwarding" should be disabled for the SVI via the appropriate sysctl nodes.

```
sysctl -w net.ipv4.conf.<ifname>.forwarding=0
sysctl -w net.ipv6.conf.<ifname>.forwarding=0
```

The following commands create a `vxlan` interface for VNI 100. Other than the VNI, The interface settings are the same for an L2VNI as they are for an L3VNI.

```
ip link add vni110 type vxlan local 100.64.0.1 dstport 4789 id 110 nolearning
ip link set vni110 master br10 addrgenmode none
ip link set vni110 type bridge_slave neigh_suppress on learning off
```

Finally, to limit a traditional bridge's broadcast domain to traffic matching specific VLAN-IDs, `vlan` subinterfaces of a host/network port need to be setup. This example shows the creation of a VLAN subinterface of "eth0" matching VID 10 with the name "eth0.10". By enslaving "eth0.10" to "br10" (instead of "eth0") we ensure that only Ethernet frames ingressing "eth0" tagged with VID 10 will be associated with the "br10" broadcast domain.

```
ip link add link eth0 name eth0.10 type vlan id 10
ip link set eth0.10 master br10
```

If you do not want to restrict the broadcast domain by VLAN-ID, you can skip the creation of the VLAN subinterfaces and directly enslave "eth0" to "br10".

```
ip link set eth0 master br10
```

This completes the necessary configuration for an L2VNI.

Displaying EVPN information

show evpn mac vni (1-16777215) detail [json]

Display detailed information about MAC addresses for a specified VNI.

show vrf [<NAME$vrf_name|all$vrf_all>] vni [json]

Displays VRF to L3VNI mapping. It also displays L3VNI associated router-mac, svi interface and vxlan interface. User can get that information as JSON format when `json` keyword at the end of cli is presented.

```
tor2# show vrf vni
VRF                                   VNI        VxLAN IF             L3-SVI       ␣
↪        State Rmac
sym_1                                 9288       vxlan21              vlan210_l3   ␣
↪        Up    21:31:36:ff:ff:20
sym_2                                 9289       vxlan21              vlan210_l3   ␣
↪        Up    21:31:36:ff:ff:20
sym_3                                 9290       vxlan21              vlan210_l3   ␣
↪        Up    21:31:36:ff:ff:20
tor2# show vrf sym_1 vni
VRF                                   VNI        VxLAN IF             L3-SVI       ␣
↪        State Rmac
sym_1                                 9288       vxlan21              vlan210_l3   ␣
↪        Up    44:38:36:ff:ff:20
```

3.9 ISIS

ISIS (Intermediate System to Intermediate System) is a routing protocol which is described in *ISO10589*, **RFC 1195**, **RFC 5308**. ISIS is an IGP (Interior Gateway Protocol). Compared with RIP, ISIS can provide scalable network support and faster convergence times like OSPF. ISIS is widely used in large networks such as ISP (Internet Service Provider) and carrier backbone networks.

3.9.1 Configuring isisd

There are no *isisd* specific options. Common options can be specified (*Common Invocation Options*) to *isisd*. *isisd* needs to acquire interface information from *zebra* in order to function. Therefore *zebra* must be running before invoking *isisd*. Also, if *zebra* is restarted then *isisd* must be too.

Like other daemons, *isisd* configuration is done in ISIS specific configuration file `isisd.conf`.

3.9.2 ISIS router

To start the ISIS process you have to specify the ISIS router. As of this writing, *isisd* does not support multiple ISIS processes.

`router isis WORD [vrf NAME]`

Enable or disable the ISIS process by specifying the ISIS domain with 'WORD'. *isisd* does not yet support multiple ISIS processes but you must specify the name of ISIS process. The ISIS process name 'WORD' is then used for interface (see command `ip router isis WORD`).

`net XX.XXXX. ... .XXX.XX`

Set/Unset network entity title (NET) provided in ISO format.

`hostname dynamic`

Enable support for dynamic hostname.

`area-password [clear | md5] <password>`

`domain-password [clear | md5] <password>`

Configure the authentication password for an area, respectively a domain, as clear text or md5 one.

`attached-bit [receive ignore | send]`

Set attached bit for inter-area traffic:

- receive If LSP received with attached bit set, create default route to neighbor
- send If L1|L2 router, set attached bit in LSP sent to L1 router

`log-adjacency-changes`

Log changes in adjacency state.

`log-pdu-drops`

Log any dropped PDUs.

`metric-style [narrow | transition | wide]`

Set old-style (ISO 10589) or new-style packet formats:

- narrow Use old style of TLVs with narrow metric
- transition Send and accept both styles of TLVs during transition
- wide Use new style of TLVs to carry wider metric. FRR uses this as a default value

advertise-high-metrics
Advertise high metric value on all interfaces to gracefully shift traffic off the router. Reference: RFC 3277

For narrow metrics, the high metric value is 63; for wide metrics, 16777215; for transition metrics, 62.

set-overload-bit
Set overload bit to avoid any transit traffic.

set-overload-bit on-startup (0-86400)
Set overload bit on startup for the specified duration, in seconds. Reference: RFC 3277

purge-originator
Enable or disable RFC 6232 purge originator identification.

lsp-mtu (128-4352)
Configure the maximum size of generated LSPs, in bytes.

advertise-passive-only
Advertise prefixes of passive interfaces only.

3.9.3 ISIS Timer

lsp-gen-interval [level-1 | level-2] (1-120)
Set minimum interval in seconds between regenerating same LSP, globally, for an area (level-1) or a domain (level-2).

lsp-refresh-interval [level-1 | level-2] (1-65235)
Set LSP refresh interval in seconds, globally, for an area (level-1) or a domain (level-2).

max-lsp-lifetime [level-1 | level-2] (360-65535)
Set LSP maximum LSP lifetime in seconds, globally, for an area (level-1) or a domain (level-2).

spf-interval [level-1 | level-2] (1-120)
Set minimum interval between consecutive SPF calculations in seconds.

3.9.4 ISIS Fast-Reroute

Unless stated otherwise, commands in this section apply to all LFA flavors (local LFA, Remote LFA and TI-LFA).

spf prefix-priority [critical | high | medium] WORD
Assign a priority to the prefixes that match the specified access-list.

By default loopback prefixes have medium priority and non-loopback prefixes have low priority.

fast-reroute priority-limit [critical | high | medium] [level-1 | level-2]
Limit LFA backup computation up to the specified prefix priority.

fast-reroute lfa tiebreaker [downstream | lowest-backup-metric | node-protecting] index (1-255) [level-
Configure a tie-breaker for multiple local LFA backups. Lower indexes are processed first.

fast-reroute load-sharing disable [level-1 | level-2]
Disable load sharing across multiple LFA backups.

fast-reroute remote-lfa prefix-list [WORD] [level-1 | level-2]
Configure a prefix-list to select eligible PQ nodes for remote LFA backups (valid for all protected interfaces).

3.9.5 ISIS region

`is-type [level-1 | level-1-2 | level-2-only]`

Define the ISIS router behavior:

- level-1 Act as a station router only
- level-1-2 Act as both a station router and an area router
- level-2-only Act as an area router only

3.9.6 ISIS interface

`<ip|ipv6> router isis WORD`

Activate ISIS adjacency on this interface. Note that the name of ISIS instance must be the same as the one used to configure the ISIS process (see command `router isis WORD`). To enable IPv4, issue `ip router isis WORD`; to enable IPv6, issue `ipv6 router isis WORD`.

`isis circuit-type [level-1 | level-1-2 | level-2]`

Configure circuit type for interface:

- level-1 Level-1 only adjacencies are formed
- level-1-2 Level-1-2 adjacencies are formed
- level-2-only Level-2 only adjacencies are formed

`isis csnp-interval (1-600) [level-1 | level-2]`

Set CSNP interval in seconds globally, for an area (level-1) or a domain (level-2).

`isis hello padding`

Add padding to IS-IS hello packets.

`isis hello padding during-adjacency-formation`

Add padding to IS-IS hello packets during adjacency formation only.

`isis hello-interval (1-600) [level-1 | level-2]`

Set Hello interval in seconds globally, for an area (level-1) or a domain (level-2).

`isis hello-multiplier (2-100) [level-1 | level-2]`

Set multiplier for Hello holding time globally, for an area (level-1) or a domain (level-2).

`isis metric [(0-255) | (0-16777215)] [level-1 | level-2]`

Set default metric value globally, for an area (level-1) or a domain (level-2). Max value depend if metric support narrow or wide value (see command *`metric-style [narrow | transition | wide]`*).

`isis network point-to-point`

Set network type to 'Point-to-Point' (broadcast by default).

`isis passive`

Configure the passive mode for this interface.

`isis password [clear | md5] <password>`

Configure the authentication password (clear or encoded text) for the interface.

`isis priority (0-127) [level-1 | level-2]`

Set priority for Designated Router election, globally, for the area (level-1) or the domain (level-2).

`isis psnp-interval (1-120) [level-1 | level-2]`

Set PSNP interval in seconds globally, for an area (level-1) or a domain (level-2).

isis three-way-handshake
: Enable or disable RFC 5303 Three-Way Handshake for P2P adjacencies. Three-Way Handshake is enabled by default.

isis fast-reroute lfa [level-1 | level-2]
: Enable per-prefix local LFA fast reroute link protection.

isis fast-reroute lfa [level-1 | level-2] exclude interface IFNAME
: Exclude an interface from the local LFA backup nexthop computation.

isis fast-reroute remote-lfa tunnel mpls-ldp [level-1 | level-2]
: Enable per-prefix Remote LFA fast reroute link protection. Note that other routers in the network need to be configured to accept LDP targeted hello messages in order for RLFA to work.

isis fast-reroute remote-lfa maximum-metric (1-16777215) [level-1 | level-2]
: Limit Remote LFA PQ node selection within the specified metric.

isis fast-reroute ti-lfa [level-1|level-2] [node-protection [link-fallback]]
: Enable per-prefix TI-LFA fast reroute link or node protection. When node protection is used, option link-fallback enables the computation and use of link-protecting LFAs for destinations unprotected by node protection.

3.9.7 Showing ISIS information

show isis [vrf <NAME|all>] summary [json]
: Show summary information about ISIS.

show isis hostname
: Show information about ISIS node.

show isis [vrf <NAME|all>] interface [detail] [IFNAME] [json]
: Show state and configuration of ISIS specified interface, or all interfaces if no interface is given with or without details.

show isis [vrf <NAME|all>] neighbor [detail] [SYSTEMID] [json]
: Show state and information of ISIS specified neighbor, or all neighbors if no system id is given with or without details.

show isis [vrf <NAME|all>] database [detail] [LSPID] [json]
: Show the ISIS database globally, for a specific LSP id without or with details.

show isis topology [level-1|level-2] [algorithm (128-255)]
: Show topology IS-IS paths to Intermediate Systems, globally, in area (level-1) or domain (level-2).

show isis route [level-1|level-2] [prefix-sid|backup] [algorithm (128-255)]
: Show the ISIS routing table, as determined by the most recent SPF calculation.

show isis fast-reroute summary [level-1|level-2]
: Show information about the number of prefixes having LFA protection, and network-wide LFA coverage.

3.9.8 Traffic Engineering

Note: IS-IS-TE supports RFC 5305 (base TE), RFC 6119 (IPv6) and RFC 7810 / 8570 (Extended Metric) with or without Multi-Topology. All Traffic Engineering information are stored in a database formally named TED. However, best acccuracy is provided without Multi-Topology due to inconsistency of Traffic Engineering Advertisement of 3rd party commercial routers when MT is enabled. At this time, FRR offers partial support for some of the routing protocol extensions that can be used with MPLS-TE. FRR does not currently support a complete RSVP-TE solution.

`mpls-te on`
Enable Traffic Engineering LSP flooding.

`mpls-te router-address <A.B.C.D>`
Configure stable IP address for MPLS-TE.

`mpls-te router-address ipv6 <X:X::X:X>`
Configure stable IPv6 address for MPLS-TE.

`mpls-te export`
Export Traffic Engineering DataBase to other daemons through the ZAPI Opaque Link State messages.

`show isis mpls-te interface`

`show isis mpls-te interface INTERFACE`
Show MPLS Traffic Engineering parameters for all or specified interface.

`show isis mpls-te router`
Show Traffic Engineering router parameters.

`show isis [vrf <NAME|all>] mpls-te database [detail|json]`

`show isis [vrf <NAME|all>] mpls-te database vertex [WORD] [detail|json]`

`show isis [vrf <NAME|all>] mpls-te database edge [A.B.C.D|X:X::X:X] [detail|json]`

`show isis [vrf <NAME|all>] mpls-te database subnet [A.B.C.D/M|X:X::X:X/M] [detail|json]`
Show Traffic Engineering Database

See also:

Traffic Engineering

3.9.9 Segment Routing

This is an EXPERIMENTAL support of Segment Routing as per RFC8667 for MPLS dataplane. It supports IPv4, IPv6 and ECMP and has been tested against Cisco & Juniper routers.

Known limitations:

- No support for level redistribution (L1 to L2 or L2 to L1)
- No support for binding SID
- No support for SRMS
- No support for SRLB
- Only one SRGB and default SPF Algorithm is supported

`segment-routing on`
Enable Segment Routing.

`segment-routing global-block (16-1048575) (16-1048575) [local-block (16-1048575) (16-1048575)]`

Set the Segment Routing Global Block i.e. the label range used by MPLS to store label in the MPLS FIB for Prefix SID. Note that the block size may not exceed 65535. Optionally sets also the Segment Routing Local Block. The negative command always unsets both.

`segment-routing node-msd (1-16)`

Set the Maximum Stack Depth supported by the router. The value depend of the MPLS dataplane. E.g. for Linux kernel, since version 4.13 the maximum value is 32.

`segment-routing prefix <A.B.C.D/M|X:X::X:X/`
`M> [algorithm (128-255)] <absolute (16-1048575)|index (0-65535) [no-php-flag|explicit-null] [n-flag-clea`

prefix. The 'no-php-flag' means NO Penultimate Hop Popping that allows SR node to request to its neighbor to not pop the label. The 'explicit-null' flag allows SR node to request to its neighbor to send IP packet with the EXPLICIT-NULL label. The 'n-flag-clear' option can be used to explicitly clear the Node flag that is set by default for Prefix-SIDs associated to loopback addresses. This option is necessary to configure Anycast-SIDs.

`show isis segment-routing node [algorithm (128-255)]`

Show detailed information about all learned Segment Routing Nodes.

3.9.10 Flex-Algos (Flex-Algo)

isisd supports some features of RFC 9350 on an MPLS Segment-Routing dataplane. The compatibility has been tested against Cisco.

IS-IS uses by default the *Shortest-Path-First* algorithm that basically calculates paths based on the shortest total metric to the destinations. Flex-Algo allows new algorithms to run in parallel to compute paths in different manners, based on metrics (IGP metric or a new type of metrics such as Traffic Engineering (TE) metric and minimum delay. . .) and constraints. New metric types are not yet implemented but constraints are already operational. Constraints can restrict paths to links with specific affinities or avoid links with specific affinities. Combinations of these are also possible.

The administrator can configure up to 128 Flex-Algos in an IS-IS area. To do so, it defines a set of Flex-Algo Definitions (FAD) which have the following characteristics:

- a numeric identifier (ID) between 128 and 255 inclusive
- **a set of constraints (basically, include or exclude a certain given set of** links, designated by a admin-group)
- the calculation type (only the *Shortest-Path-First* is currently supported)
- the metric type (only the IGP inherited metric type is currently supported)
- some additional flags (not supported for the moment).

A subset of routers advertises the Flex-Algo Definitions (FAD) to the other routers within an area. In order to use a common set of FADs, each router runs a FAD election process for each locally configured algorithm, using the following rules:

- **If a locally configured FAD is not advertised to the area, the router does not** participate in the particular flex algorithm.
- **If a given flex algorithm is running, the participation in this particular** flex algorithm stops when its advertisements are over.
- **A router includes its own FAD in the election process if and only if it is** advertised to the other routers.
- If only one router advertises the FAD, the FAD is elected.
- **If several FADs are advertised with different priorities, the one with the** highest priority value is selected.
- **If there are multiple advertisements of the FAD with the same highest** priority, the FAD of the router with the highest IS-IS system-ID is selected.

Routers only use the specifications of the elected FAD regardless of the locally configured definitions. If a router does not support one of the FAD characteristics, it stops participating in the Flex-Algo.

For each running Flex-Algo, the Segment-Routing SIDs must be configured with values unique to the algorithm. It allows routers to identify which flex algorithm they must use for a given packet.

The following commands configure Flex-Algo at the 'router isis' configuration level. Segment-Routing prefixes must be configured for the Flex-Algo.

`flexible-algorithm (128-255)`
: Add a Flex-Algo Definition (FAD) and enter the FAD configuration level. The algorithm ID value is in the range of 128 to 255 inclusive.

`no flexible-algorithm (128-255)`
: Unconfigure a Flex-Algo Definition.

`affinity-map NAME bit-position (0-255)`
: Add the specified 'affinity-map'. Affinity-map definitions are used in FADs and in interfaces admin-group definition.

  Affinity-maps format in advertisement TLVs use the extended admin-group format defined in the RFC7308 section 2.2. The extended admin-group uses a 256 bits field. If an affinity-map is set, the bit at the extended admin-group 'bit-position' is set 1, else it is set to 0.

The following commands configure Flex-Algo at the 'router isis' and 'flexible-algorithm (128-255)' configuration level.

`advertise-definition`
: Advertise the current FAD to other IS-IS routers by using specific IS-IS TLVs. By default, the definition is is not shared with other routers.

A router can advertise a FAD without participating in the Flex-Algo.

`priority (0-255)`
: Set the specified 'priority' in the current FAD advertisements .

`metric-type [igp|te|delay]`
: Set the 'metric-type' for the current FAD. 'igp' is the default value and refers to the classic 'Shortest-Path-First' algorithm. If the 'te' or the 'delay' metric is selected, the value is advertised but the flex algorithm is disabled locally because these types are not currently supported.

`no metric-type`
: Reset the 'metric-type' to the default 'igp' metric.

`affinity exclude-any NAME`
: Add the specified affinity to the list of exclude-any affinities. The Flex-Algo will compute paths that exclude the segments with any of the specified affinities.

`no affinity exclude-any NAME`
: Remove the specified affinity to the list of exclude-any affinities.

`affinity include-all NAME`
: Add the specified affinity to the list of include-all affinities. The Flex-Algo will compute paths that include the segments with all the specified affinities.

`no affinity include-all NAME`
: Remove the specified affinity to the list of include-all affinities.

`affinity include-any NAME`
: Add the specified affinity to the list of include-any affinities. The Flex-Algo will compute paths that include the segments with any of the specified affinities.

`no affinity include-any NAME`
  Remove the specified affinity to the list of include-any affinities.

The following commands configure Flex-Algo at the 'interface' configuration level.

`isis affinity flex-algo NAME`
  Add the specified affinity to the interface.

`no isis affinity flex-algo NAME`
  Remove the specified affinity from the interface.

The following command show Flex-Algo information:

`show isis flex-algo [(128-255)]`
  Show information about the elected FADs

'show isis route', 'show isis topology' and 'show isis segment-routing node' includes an 'algorithm (128-255)' optional argument. See *Showing ISIS information* and *Segment Routing*.

3.9.11 Debugging ISIS

`debug isis adj-packets`
  IS-IS Adjacency related packets.

`debug isis checksum-errors`
  IS-IS LSP checksum errors.

`debug isis events`
  IS-IS Events.

`debug isis local-updates`
  IS-IS local update packets.

`debug isis packet-dump`
  IS-IS packet dump.

`debug isis protocol-errors`
  IS-IS LSP protocol errors.

`debug isis route-events`
  IS-IS Route related events.

`debug isis snp-packets`
  IS-IS CSNP/PSNP packets.

`debug isis spf-events`

`debug isis spf-statistics`

`debug isis spf-triggers`
  IS-IS Shortest Path First Events, Timing and Statistic Data and triggering events.

`debug isis update-packets`
  Update related packets.

`debug isis te-events`
  IS-IS Traffic Engineering events

`debug isis sr-events`
  IS-IS Segment Routing events.

`debug isis lfa`
  IS-IS LFA events.

show debugging isis
Print which ISIS debug level is activate.

3.9.12 ISIS Configuration Examples

A simple example, with MD5 authentication enabled:

```
!
interface eth0
 ip router isis FOO
 isis network point-to-point
 isis circuit-type level-2-only
!
router isis FOO
net 47.0023.0000.0000.0000.0000.0000.0000.1900.0004.00
 metric-style wide
 is-type level-2-only
```

A Traffic Engineering configuration, with Inter-ASv2 support.

First, the `zebra.conf` part:

```
hostname HOSTNAME
password PASSWORD
log file /var/log/zebra.log
!
interface eth0
 ip address 10.2.2.2/24
 link-params
  max-bw 1.25e+07
  max-rsv-bw 1.25e+06
  unrsv-bw 0 1.25e+06
  unrsv-bw 1 1.25e+06
  unrsv-bw 2 1.25e+06
  unrsv-bw 3 1.25e+06
  unrsv-bw 4 1.25e+06
  unrsv-bw 5 1.25e+06
  unrsv-bw 6 1.25e+06
  unrsv-bw 7 1.25e+06
  admin-grp 0xab
!
interface eth1
 ip address 10.1.1.1/24
 link-params
  enable
  metric 100
  max-bw 1.25e+07
  max-rsv-bw 1.25e+06
  unrsv-bw 0 1.25e+06
  unrsv-bw 1 1.25e+06
  unrsv-bw 2 1.25e+06
  unrsv-bw 3 1.25e+06
  unrsv-bw 4 1.25e+06
```

(continues on next page)

(continued from previous page)

```
  unrsv-bw 5 1.25e+06
  unrsv-bw 6 1.25e+06
  unrsv-bw 7 1.25e+06
  neighbor 10.1.1.2 as 65000
```

Then the `isisd.conf` itself:

```
hostname HOSTNAME
password PASSWORD
log file /var/log/isisd.log
!
!
interface eth0
 ip router isis FOO
!
interface eth1
 ip router isis FOO
!
!
router isis FOO
 isis net 47.0023.0000.0000.0000.0000.0000.0000.1900.0004.00
  mpls-te on
  mpls-te router-address 10.1.1.1
!
line vty
```

A Segment Routing configuration, with IPv4, IPv6, SRGB and MSD configuration.

```
hostname HOSTNAME
password PASSWORD
log file /var/log/isisd.log
!
!
interface eth0
 ip router isis SR
 isis network point-to-point
!
interface eth1
 ip router isis SR
!
!
router isis SR
 net 49.0000.0000.0000.0001.00
 is-type level-1
 topology ipv6-unicast
 lsp-gen-interval 2
 segment-routing on
 segment-routing node-msd 8
 segment-routing prefix 10.1.1.1/32 index 100 explicit-null
 segment-routing prefix 2001:db8:1000::1/128 index 101 explicit-null
!
```

3.9.13 ISIS Vrf Configuration Examples

A simple vrf example:

```
!
interface eth0 vrf RED
 ip router isis FOO vrf RED
 isis network point-to-point
 isis circuit-type level-2-only
!
router isis FOO vrf RED
 net 47.0023.0000.0000.0000.0000.0000.0000.1900.0004.00
 metric-style wide
 is-type level-2-only
```

3.10 NHRP

nhrpd is an implementation of the NHRP (Next Hop Routing Protocol). NHRP is described in **RFC 2332**.

NHRP is used to improve the efficiency of routing computer network traffic over NBMA (Non-Broadcast, Multiple Access) networks. NHRP provides an ARP-like solution that allows a system to dynamically learn the NBMA address of the other systems that are part of that network, allowing these systems to directly communicate without requiring traffic to use an intermediate hop.

NHRP is a client-server protocol. The server side is called the NHS (Next Hop Server) or the hub, while a client is referred to as the NHC (Next Hop Client) or the spoke. When a node is configured as an NHC, it registers its address with the NHS which keeps track of all registered spokes. An NHC client can then query the addresses of other clients from NHS allowing all spokes to communicate directly with each other.

Cisco Dynamic Multipoint VPN (DMVPN) is based on NHRP, and frr nhrpd implements this scenario.

3.10.1 Routing Design

nhrpd never handles routing of prefixes itself. You need to run some real routing protocol (e.g. BGP) to advertise routes over the tunnels. What nhrpd does it establishes 'shortcut routes' that optimizes the routing protocol to avoid going through extra nodes in NBMA GRE mesh.

nhrpd does route NHRP domain addresses individually using per-host prefixes. This is similar to Cisco FlexVPN; but in contrast to opennhrp which uses a generic subnet route.

To create NBMA GRE tunnel you might use the following (Linux terminal commands):

```
ip tunnel add gre1 mode gre key 42 ttl 64
ip addr add 10.255.255.2/32 dev gre1
ip link set gre1 up
```

Note that the IP-address is assigned as host prefix to gre1. nhrpd will automatically create additional host routes pointing to gre1 when a connection with these hosts is established.

The gre1 subnet prefix should be announced by routing protocol from the hub nodes (e.g. BGP 'network' announce). This allows the routing protocol to decide which is the closest hub and determine the relay hub on prefix basis when direct tunnel is not established.

nhrpd will redistribute directly connected neighbors to zebra. Within hub nodes, these routes should be internally redistributed using some routing protocol (e.g. iBGP) to allow hubs to be able to relay all traffic.

This can be achieved in hubs with the following bgp configuration (network command defines the GRE subnet):

```
router bgp 65555
 address-family ipv4 unicast
   network 172.16.0.0/16
   redistribute nhrp
 exit-address-family
```

3.10.2 Configuring NHRP

`ip nhrp holdtime (1-65000)`
Holdtime is the number of seconds that have to pass before stopping to advertise an NHRP NBMA address as valid. It also controls how often NHRP registration requests are sent. By default registrations are sent every one third of the holdtime.

`ip nhrp map A.B.C.D|X:X::X:X A.B.C.D|local`
Map an IP address of a station to the station's NBMA address.

`ip nhrp network-id (1-4294967295)`
Enable NHRP on this interface and set the interface's network ID. The network ID is used to allow creating multiple nhrp domains on a router when multiple interfaces are configured on the router. Interfaces configured with the same ID are part of the same logical NBMA network. The ID is a local only parameter and is not sent to other NHRP nodes and so IDs on different nodes do not need to match. When NHRP packets are received on an interface they are assigned to the local NHRP domain for that interface.

`ip nhrp nhs A.B.C.D nbma A.B.C.D|FQDN`
Configure the Next Hop Server address and its NBMA address.

`ip nhrp nhs dynamic nbma A.B.C.D`
Configure the Next Hop Server to have a dynamic address and set its NBMA address.

`ip nhrp registration no-unique`
Allow the client to not set the unique flag in the NHRP packets. This is useful when a station has a dynamic IP address that could change over time.

`ip nhrp shortcut`
Enable shortcut (spoke-to-spoke) tunnels to allow NHC to talk to each others directly after establishing a connection without going through the hub.

`ip nhrp mtu`
Configure NHRP advertised MTU.

3.10.3 Hub Functionality

In addition to routing nhrp redistributed host prefixes, the hub nodes are also responsible to send NHRP Traffic Indication messages that trigger creation of the shortcut tunnels.

nhrpd sends Traffic Indication messages based on network traffic captured using NFLOG. Typically you want to send Traffic Indications for network traffic that is routed from gre1 back to gre1 in rate limited manner. This can be achieved with the following iptables rule.

```
iptables -A FORWARD -i gre1 -o gre1 \\
    -m hashlimit --hashlimit-upto 4/minute --hashlimit-burst 1 \\
    --hashlimit-mode srcip,dstip --hashlimit-srcmask 24 --hashlimit-dstmask 24 \\
    --hashlimit-name loglimit-0 -j NFLOG --nflog-group 1 --nflog-range 128
```

You can fine tune the src/dstmask according to the prefix lengths you announce internal, add additional IP range matches, or rate limitation if needed. However, the above should be good in most cases.

This kernel NFLOG target's nflog-group is configured in global nhrp config with:

nhrp nflog-group (1-65535)

To start sending these traffic notices out from hubs, use the nhrp per-interface directive:

ip nhrp redirect

This enable redirect replies on the NHS similar to ICMP redirects except this is managed by the nhrp protocol. This setting allows spokes to communicate with each others directly.

3.10.4 Integration with IKE

nhrpd needs tight integration with IKE daemon for various reasons. Currently only strongSwan is supported as IKE daemon.

nhrpd connects to strongSwan using VICI protocol based on UNIX socket which can be configured using the command below (default to /var/run/charon.vici).

strongSwan currently needs few patches applied. Please check out the original patches at: https://git-old.alpinelinux.org/user/tteras/strongswan/

Actively maintained patches are also available at: https://gitlab.alpinelinux.org/alpine/aports/-/tree/master/main/strongswan

3.10.5 Multicast Functionality

nhrpd can be configured to forward multicast packets, allowing routing protocols that use multicast (such as OSPF) to be supported in the DMVPN network.

This support requires an iptables NFLOG rule to allow nhrpd to intercept multicast packets. A second iptables rule is also usually used to drop the original multicast packet.

```
iptables -A OUTPUT -d 224.0.0.0/24 -o gre1 -j NFLOG --nflog-group 2
iptables -A OUTPUT -d 224.0.0.0/24 -o gre1 -j DROP
```

nhrp multicast-nflog-group (1-65535)
: Sets the nflog group that nhrpd will listen on for multicast packets. This value must match the nflog-group value set in the iptables rule.

ip nhrp map multicast A.B.C.D|X:X::X:X A.B.C.D|dynamic
: Sends multicast packets to the specified NBMA address. If dynamic is specified then destination NBMA address (or addresses) are learnt dynamically.

3.10.6 NHRP Events

nhrp event socket SOCKET
Configure the Unix path for the event socket.

3.10.7 Show NHRP

show [ip|ipv6] nhrp cache [json]
Dump the cache entries.

show [ip|ipv6] nhrp opennhrp [json]
Dump the cache entries with opennhrp format.

show [ip|ipv6] nhrp nhs [json]
Dump the hub context.

show dmvpn [json]
Dump the security contexts.

3.10.8 Configuration Example

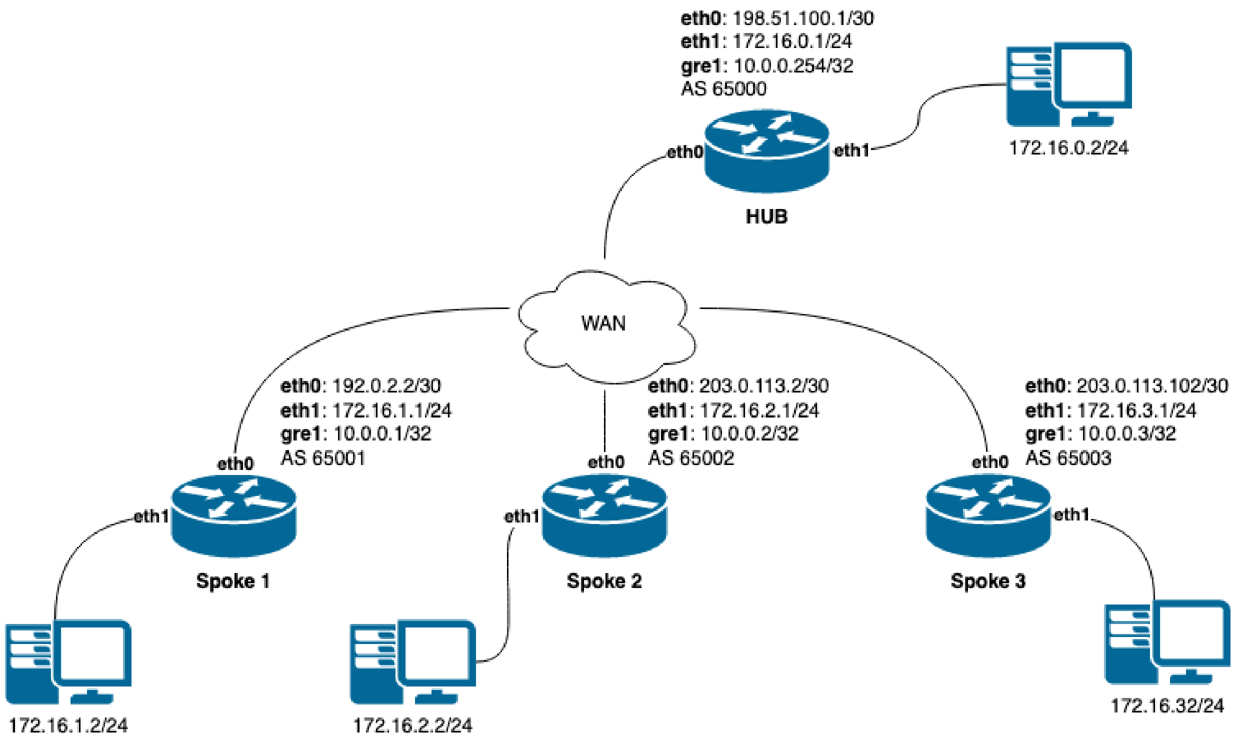


Fig. 5: image

IPSec configurration example

This changes required on all nodes as HUB and Spokes.

ipsec.conf file

```
config setup
conn dmvpn
    authby=secret
    auto=add
    keyexchange=ikev2
    ike=aes256-aes256-sha256-modp2048
    esp=aes256-aes256-sha256-modp2048
    dpdaction=clear
    dpddelay=300s
    left=%any
    leftid=%any
    right=%any
    rightid=%any
    leftprotoport=gre
    rightprotoport=gre
    type=transport
    keyingtries=%forever
```

ipsec.secrets file

```
%any : PSK "some_s3cret!"
```

HUB configuration example

Creating gre interface

```
ip tunnel add gre1 mode gre key 42 ttl 64
ip addr add 10.0.0.254/32 dev gre1
ip link set gre1 up
```

Adding iptables rules to provide possibility shortcut tunnels and connect spokes directly

```
iptables -A FORWARD -i gre1 -o gre1 \\
    -m hashlimit --hashlimit-upto 4/minute --hashlimit-burst 1 \\
    --hashlimit-mode srcip,dstip --hashlimit-srcmask 24 --hashlimit-dstmask 24 \\
    --hashlimit-name loglimit-0 -j NFLOG --nflog-group 1 --nflog-range 128
```

FRR config on HUB

```
nhrp nflog-group 1
!
interface gre1
 description DMVPN Tunnel Interface
 ip address 10.0.0.254/32
 ip nhrp network-id 1
 ip nhrp redirect
 ip nhrp registration no-unique
```

(continues on next page)

(continued from previous page)

```
 ip nhrp shortcut
 tunnel protection vici profile dmvpn
 tunnel source eth0
 !
 router bgp 65000
  bgp router-id 10.0.0.254
  no bgp ebgp-requires-policy
  neighbor SPOKES peer-group
  neighbor SPOKES disable-connected-check
  neighbor 10.0.0.1 remote-as 65001
  neighbor 10.0.0.1 peer-group SPOKES
  neighbor 10.0.0.2 remote-as 65002
  neighbor 10.0.0.2 peer-group SPOKES
  neighbor 10.0.0.3 remote-as 65003
  neighbor 10.0.0.3 peer-group SPOKES
  !
  address-family ipv4 unicast
   network 172.16.0.0/24
   redistribute nhrp
  exit-address-family
```

Spoke1 configuration

Creating gre interface

```
ip tunnel add gre1 mode gre key 42 ttl 64
ip addr add 10.0.0.1/32 dev gre1
ip link set gre1 up
```

FRR config on Spoke1

```
interface gre1
 description DMVPN Tunnel Interface
 ip address 10.0.0.1/32
 ip nhrp network-id 1
 ip nhrp nhs dynamic nbma 198.51.100.1
 ip nhrp redirect
 ip nhrp registration no-unique
 ip nhrp shortcut
 no link-detect
 tunnel protection vici profile dmvpn
 tunnel source eth0
!
router bgp 65001
 no bgp ebgp-requires-policy
 neighbor 10.0.0.254 remote-as 65000
 neighbor 10.0.0.254 disable-connected-check
 !
 address-family ipv4 unicast
  network 172.16.1.0/24
 exit-address-family
```

Spoke2 configuration

Creating gre interface

```
ip tunnel add gre1 mode gre key 42 ttl 64
ip addr add 10.0.0.1/32 dev gre1
ip link set gre1 up
```

FRR config on Spoke2

```
interface gre1
 description DMVPN Tunnel Interface
 ip address 10.0.0.2/32
 ip nhrp network-id 1
 ip nhrp nhs dynamic nbma 198.51.100.1
 ip nhrp redirect
 ip nhrp registration no-unique
 ip nhrp shortcut
 no link-detect
 tunnel protection vici profile dmvpn
 tunnel source eth0
!
router bgp 65002
 no bgp ebgp-requires-policy
 neighbor 10.0.0.254 remote-as 65000
 neighbor 10.0.0.254 disable-connected-check
 !
 address-family ipv4 unicast
  network 172.16.2.0/24
 exit-address-family
```

Spoke3 configuration

Creating gre interface

```
ip tunnel add gre1 mode gre key 42 ttl 64
ip addr add 10.0.0.3/32 dev gre1
ip link set gre1 up
```

FRR config on Spoke3

```
interface gre1
 description DMVPN Tunnel Interface
 ip address 10.0.0.3/32
 ip nhrp network-id 1
 ip nhrp nhs dynamic nbma 198.51.100.1
 ip nhrp redirect
 ip nhrp registration no-unique
 ip nhrp shortcut
 no link-detect
 tunnel protection vici profile dmvpn
 tunnel source eth0
!
```

(continues on next page)

(continued from previous page)

```
router bgp 65003
 no bgp ebgp-requires-policy
 neighbor 10.0.0.254 remote-as 65000
 neighbor 10.0.0.254 disable-connected-check
 !
 address-family ipv4 unicast
  network 172.16.3.0/24
 exit-address-family
```

3.11 OSPFv2

OSPF (Open Shortest Path First) version 2 is a routing protocol which is described in **RFC 2328**. OSPF is an IGP. Compared with RIP, OSPF can provide scalable network support and faster convergence times. OSPF is widely used in large networks such as ISP backbone and enterprise networks.

3.11.1 OSPF Fundamentals

OSPF is, mostly, a link-state routing protocol. In contrast to *distance-vector* protocols, such as RIP or BGP, where routers describe available *paths* (i.e. routes) to each other, in *link-state* protocols routers instead describe the state of their links to their immediate neighbouring routers.

Each router describes their link-state information in a message known as an LSA (Link State Advertisement), which is then propagated through to all other routers in a link-state routing domain, by a process called *flooding*. Each router thus builds up an LSDB (Link State Database) of all the link-state messages. From this collection of LSAs in the LSDB, each router can then calculate the shortest path to any other router, based on some common metric, by using an algorithm such as Edsger Dijkstra's SPF (Shortest Path First) algorithm.

By describing connectivity of a network in this way, in terms of routers and links rather than in terms of the paths through a network, a link-state protocol can use less bandwidth and converge more quickly than other protocols. A link-state protocol need distribute only one link-state message throughout the link-state domain when a link on any single given router changes state, in order for all routers to reconverge on the best paths through the network. In contrast, distance vector protocols can require a progression of different path update messages from a series of different routers in order to converge.

The disadvantage to a link-state protocol is that the process of computing the best paths can be relatively intensive when compared to distance-vector protocols, in which near to no computation need be done other than (potentially) select between multiple routes. This overhead is mostly negligible for modern embedded CPUs, even for networks with thousands of nodes. The primary scaling overhead lies more in coping with the ever greater frequency of LSA updates as the size of a link-state area increases, in managing the LSDB and required flooding.

This section aims to give a distilled, but accurate, description of the more important workings of OSPF which an administrator may need to know to be able best configure and trouble-shoot OSPF.

OSPF Mechanisms

OSPF defines a range of mechanisms, concerned with detecting, describing and propagating state through a network. These mechanisms will nearly all be covered in greater detail further on. They may be broadly classed as:

The Hello Protocol

The OSPF Hello protocol allows OSPF to quickly detect changes in two-way reachability between routers on a link. OSPF can additionally avail of other sources of reachability information, such as link-state information provided by hardware, or through dedicated reachability protocols such as BFD.

OSPF also uses the Hello protocol to propagate certain state between routers sharing a link, for example:

- Hello protocol configured state, such as the dead-interval.
- Router priority, for DR/BDR election.
- DR/BDR election results.
- Any optional capabilities supported by each router.

The Hello protocol is comparatively trivial and will not be explored in more detail.

LSAs

At the heart of OSPF are LSA messages. Despite the name, some LSA s do not, strictly speaking, describe link-state information. Common LSA s describe information such as:

- Routers, in terms of their links.
- Networks, in terms of attached routers.
- Routes, external to a link-state domain:

  External Routes Routes entirely external to OSPF. Routers originating such routes are known as ASBR (Autonomous-System Border Router) routers.

  Summary Routes Routes which summarise routing information relating to OSPF areas external to the OSPF link-state area at hand, originated by ABR (Area Boundary Router) routers.

LSA Flooding

OSPF defines several related mechanisms, used to manage synchronisation of LSDB s between neighbours as neighbours form adjacencies and the propagation, or *flooding* of new or updated LSA s.

Areas

OSPF provides for the protocol to be broken up into multiple smaller and independent link-state areas. Each area must be connected to a common backbone area by an ABR. These ABR routers are responsible for summarising the link-state routing information of an area into *Summary LSAs*, possibly in a condensed (i.e. aggregated) form, and then originating these summaries into all other areas the ABR is connected to.

Note that only summaries and external routes are passed between areas. As these describe *paths*, rather than any router link-states, routing between areas hence is by *distance-vector*, **not** link-state.

OSPF LSAs

The core objects in OSPF are LSA s. Everything else in OSPF revolves around detecting what to describe in LSAs, when to update them, how to flood them throughout a network and how to calculate routes from them.

There are a variety of different LSA s, for purposes such as describing actual link-state information, describing paths (i.e. routes), describing bandwidth usage of links for TE (Traffic Engineering) purposes, and even arbitrary data by way of *Opaque* LSA s.

LSA Header

All LSAs share a common header with the following information:

- Type

  Different types of LSA s describe different things in OSPF. Types include:

  - Router LSA
  - Network LSA
  - Network Summary LSA
  - Router Summary LSA
  - AS-External LSA

  The specifics of the different types of LSA are examined below.

- Advertising Router

  The Router ID of the router originating the LSA.

See also:

ospf router-id A.B.C.D.

- LSA ID

  The ID of the LSA, which is typically derived in some way from the information the LSA describes, e.g. a Router LSA uses the Router ID as the LSA ID, a Network LSA will have the IP address of the DR as its LSA ID.

  The combination of the Type, ID and Advertising Router ID must uniquely identify the LSA. There can however be multiple instances of an LSA with the same Type, LSA ID and Advertising Router ID, see *sequence number*.

- Age

  A number to allow stale LSA s to, eventually, be purged by routers from their LSDB s.

  The value nominally is one of seconds. An age of 3600, i.e. 1 hour, is called the *MaxAge*. MaxAge LSAs are ignored in routing calculations. LSAs must be periodically refreshed by their Advertising Router before reaching MaxAge if they are to remain valid.

  Routers may deliberately flood LSAs with the age artificially set to 3600 to indicate an LSA is no longer valid. This is called *flushing* of an LSA.

  It is not abnormal to see stale LSAs in the LSDB, this can occur where a router has shutdown without flushing its LSA(s), e.g. where it has become disconnected from the network. Such LSAs do little harm.

- Sequence Number

  A number used to distinguish newer instances of an LSA from older instances.

Link-State LSAs

Of all the various kinds of LSA s, just two types comprise the actual link-state part of OSPF, Router LSA s and Network LSA s. These LSA types are absolutely core to the protocol.

Instances of these LSAs are specific to the link-state area in which they are originated. Routes calculated from these two LSA types are called *intra-area routes*.

- Router LSA

  Each OSPF Router must originate a router LSA to describe itself. In it, the router lists each of its OSPF enabled interfaces, for the given link-state area, in terms of:

  Cost The output cost of that interface, scaled inversely to some commonly known reference value, `auto-cost reference-bandwidth (1-4294967)`.

  Link Type Transit Network

  A link to a multi-access network, on which the router has at least one Full adjacency with another router.

  PtP (Point-to-Point) A link to a single remote router, with a Full adjacency. No DR (Designated Router) is elected on such links; no network LSA is originated for such a link.

  Stub A link with no adjacent neighbours, or a host route.

  - Link ID and Data

    These values depend on the Link Type:

Link Type	Link ID	Link Data
Transit	Link IP address of the DR	Interface IP address
Point-to-Point	Router ID of the remote router	Local interface IP address, or the IFINDEX (MIB-II interface index) for unnumbered links
Stub	IP address	Subnet Mask

    Links on a router may be listed multiple times in the Router LSA, e.g. a PtP interface on which OSPF is enabled must *always* be described by a Stub link in the Router LSA, in addition to being listed as PtP link in the Router LSA if the adjacency with the remote router is Full.

    Stub links may also be used as a way to describe links on which OSPF is *not* spoken, known as *passive interfaces*, see `ip ospf passive [A.B.C.D]`.

- Network LSA

  On multi-access links (e.g. ethernets, certain kinds of ATM and X.25 configurations), routers elect a DR. The DR is responsible for originating a Network LSA, which helps reduce the information needed to describe multi-access networks with multiple routers attached. The DR also acts as a hub for the flooding of LSA s on that link, thus reducing flooding overheads.

  The contents of the Network LSA describes the:

  - Subnet Mask

    As the LSA ID of a Network LSA must be the IP address of the DR, the Subnet Mask together with the LSA ID gives you the network address.

  - Attached Routers

    Each router fully-adjacent with the DR is listed in the LSA, by their Router-ID. This allows the corresponding Router LSA s to be easily retrieved from the LSDB.

Summary of Link State LSAs:

LSA Type	LSA ID	LSA Data Describes
Router LSA	Router ID	The OSPF enabled links of the router, within a specific link-state area.
Network LSA	The IP address of the DR for the network	The subnet mask of the network and the Router IDs of all routers on the network

With an LSDB composed of just these two types of LSA, it is possible to construct a directed graph of the connectivity between all routers and networks in a given OSPF link-state area. So, not surprisingly, when OSPF routers build updated routing tables, the first stage of SPF calculation concerns itself only with these two LSA types.

Link-State LSA Examples

The example below shows two LSA s, both originated by the same router (Router ID 192.168.0.49) and with the same LSA ID (192.168.0.49), but of different LSA types.

The first LSA being the router LSA describing 192.168.0.49's links: 2 links to multi-access networks with fully-adjacent neighbours (i.e. Transit links) and 1 being a Stub link (no adjacent neighbours).

The second LSA being a Network LSA, for which 192.168.0.49 is the DR, listing the Router IDs of 4 routers on that network which are fully adjacent with 192.168.0.49.

```
# show ip ospf database router 192.168.0.49

       OSPF Router with ID (192.168.0.53)

                Router Link States (Area 0.0.0.0)

  LS age: 38
  Options: 0x2  : *|-|-|-|-|-|E|*
  LS Flags: 0x6
  Flags: 0x2 : ASBR
  LS Type: router-LSA
  Link State ID: 192.168.0.49
  Advertising Router: 192.168.0.49
  LS Seq Number: 80000f90
  Checksum: 0x518b
  Length: 60
   Number of Links: 3

    Link connected to: a Transit Network
     (Link ID) Designated Router address: 192.168.1.3
     (Link Data) Router Interface address: 192.168.1.3
      Number of TOS metrics: 0
       TOS 0 Metric: 10

    Link connected to: a Transit Network
     (Link ID) Designated Router address: 192.168.0.49
     (Link Data) Router Interface address: 192.168.0.49
      Number of TOS metrics: 0
       TOS 0 Metric: 10
```

(continues on next page)

(continued from previous page)

```
    Link connected to: Stub Network
     (Link ID) Net: 192.168.3.190
     (Link Data) Network Mask: 255.255.255.255
      Number of TOS metrics: 0
       TOS 0 Metric: 39063
# show ip ospf database network 192.168.0.49

       OSPF Router with ID (192.168.0.53)

                Net Link States (Area 0.0.0.0)

  LS age: 285
  Options: 0x2  : *|-|-|-|-|-|E|*
  LS Flags: 0x6
  LS Type: network-LSA
  Link State ID: 192.168.0.49 (address of Designated Router)
  Advertising Router: 192.168.0.49
  LS Seq Number: 80000074
  Checksum: 0x0103
  Length: 40
  Network Mask: /29
        Attached Router: 192.168.0.49
        Attached Router: 192.168.0.52
        Attached Router: 192.168.0.53
        Attached Router: 192.168.0.54
```

Note that from one LSA, you can find the other. E.g. Given the Network-LSA you have a list of Router IDs on that network, from which you can then look up, in the local LSDB, the matching Router LSA. From that Router-LSA you may (potentially) find links to other Transit networks and Routers IDs which can be used to lookup the corresponding Router or Network LSA. And in that fashion, one can find all the Routers and Networks reachable from that starting LSA.

Given the Router LSA instead, you have the IP address of the DR of any attached transit links. Network LSAs will have that IP as their LSA ID, so you can then look up that Network LSA and from that find all the attached routers on that link, leading potentially to more links and Network and Router LSAs, etc. etc.

From just the above two LSA s, one can already see the following partial topology:

```
------------------------ Network: ......
            |            Designated Router IP: 192.168.1.3
            |
      IP: 192.168.1.3
       (transit link)
        (cost: 10)
   Router ID: 192.168.0.49(stub)---------- IP: 192.168.3.190/32
        (cost: 10)        (cost: 39063)
       (transit link)
      IP: 192.168.0.49
            |
            |
------------------------------ Network: 192.168.0.48/29
  |        |           |       Designated Router IP: 192.168.0.49
  |        |           |
```

(continues on next page)

(continued from previous page)

```
  |        |     Router ID: 192.168.0.54
  |        |
  |   Router ID: 192.168.0.53
  |
Router ID: 192.168.0.52
```

Note the Router IDs, though they look like IP addresses and often are IP addresses, are not strictly speaking IP addresses, nor need they be reachable addresses (though, OSPF will calculate routes to Router IDs).

External LSAs

External, or "Type 5", LSA s describe routing information which is entirely external to OSPF, and is "injected" into OSPF. Such routing information may have come from another routing protocol, such as RIP or BGP, they may represent static routes or they may represent a default route.

An OSPF router which originates External LSA s is known as an ASBR. Unlike the link-state LSA s, and most other LSA s, which are flooded only within the area in which they originate, External LSA s are flooded through-out the OSPF network to all areas capable of carrying External LSA s (*Areas*).

Routes internal to OSPF (intra-area or inter-area) are always preferred over external routes.

The External LSA describes the following:

IP Network number The IP Network number of the route is described by the LSA ID field.

IP Network Mask The body of the External LSA describes the IP Network Mask of the route. This, together with the LSA ID, describes the prefix of the IP route concerned.

Metric The cost of the External Route. This cost may be an OSPF cost (also known as a "Type 1" metric), i.e. equivalent to the normal OSPF costs, or an externally derived cost ("Type 2" metric) which is not comparable to OSPF costs and always considered larger than any OSPF cost. Where there are both Type 1 and 2 External routes for a route, the Type 1 is always preferred.

Forwarding Address The address of the router to forward packets to for the route. This may be, and usually is, left as 0 to specify that the ASBR originating the External LSA should be used. There must be an internal OSPF route to the forwarding address, for the forwarding address to be usable.

Tag An arbitrary 4-bytes of data, not interpreted by OSPF, which may carry whatever information about the route which OSPF speakers desire.

AS External LSA Example

To illustrate, below is an example of an External LSA in the LSDB of an OSPF router. It describes a route to the IP prefix of 192.168.165.0/24, originated by the ASBR with Router-ID 192.168.0.49. The metric of 20 is external to OSPF. The forwarding address is 0, so the route should forward to the originating ASBR if selected.

```
# show ip ospf database external 192.168.165.0
  LS age: 995
  Options: 0x2  : *|-|-|-|-|-|E|*
  LS Flags: 0x9
  LS Type: AS-external-LSA
  Link State ID: 192.168.165.0 (External Network Number)
  Advertising Router: 192.168.0.49
  LS Seq Number: 800001d8
```

(continues on next page)

(continued from previous page)

```
  Checksum: 0xea27
  Length: 36
  Network Mask: /24
        Metric Type: 2 (Larger than any link state path)
        TOS: 0
        Metric: 20
        Forward Address: 0.0.0.0
        External Route Tag: 0
```

We can add this to our partial topology from above, which now looks like::

```
--------------------- Network: ......
         |            Designated Router IP: 192.168.1.3
         |
   IP: 192.168.1.3      /---- External route: 192.168.165.0/24
    (transit link)     /                Cost: 20 (External metric)
     (cost: 10)       /
Router ID: 192.168.0.49(stub)---------- IP: 192.168.3.190/32
     (cost: 10)        (cost: 39063)
    (transit link)
   IP: 192.168.0.49
         |
         |
------------------------------ Network: 192.168.0.48/29
  |        |           |       Designated Router IP: 192.168.0.49
  |        |           |
  |        |     Router ID: 192.168.0.54
  |        |
  |   Router ID: 192.168.0.53
  |
Router ID: 192.168.0.52
```

Summary LSAs

Summary LSAs are created by ABR s to summarise the destinations available within one area to other areas. These LSAs may describe IP networks, potentially in aggregated form, or ASBR routers.

3.11.2 Configuring OSPF

ospfd accepts all *Common Invocation Options*.

-n, --instance
Specify the instance number for this invocation of *ospfd*.

-a, --apiserver
Enable the OSPF API server. This is required to use `ospfclient`.

ospfd must acquire interface information from *zebra* in order to function. Therefore *zebra* must be running before invoking *ospfd*. Also, if *zebra* is restarted then *ospfd* must be too.

Like other daemons, *ospfd* configuration is done in OSPF specific configuration file `ospfd.conf` when the integrated config is not used.

Multi-instance Support

OSPF supports multiple instances. Each instance is identified by a positive nonzero integer that must be provided when adding configuration items specific to that instance. Enabling instances is done with `/etc/frr/daemons` in the following manner:

```
...
ospfd=yes
ospfd_instances=1,5,6
...
```

The `ospfd_instances` variable controls which instances are started and what their IDs are. In this example, after starting FRR you should see the following processes:

```
# ps -ef | grep "ospfd"
frr      11816     1  0 17:30 ?        00:00:00 /usr/lib/frr/ospfd --daemon -A 127.0.0.1␣
↪-n 1
frr      11822     1  0 17:30 ?        00:00:00 /usr/lib/frr/ospfd --daemon -A 127.0.0.1␣
↪-n 2
frr      11828     1  0 17:30 ?        00:00:00 /usr/lib/frr/ospfd --daemon -A 127.0.0.1␣
↪-n 3
```

The instance number should be specified in the config when addressing a particular instance:

```
router ospf 5
   ospf router-id 1.2.3.4
   area 0.0.0.0 authentication message-digest
   ...
```

Routers

To start OSPF process you have to specify the OSPF router.

`router ospf [{(1-65535)|vrf NAME}]`

Enable or disable the OSPF process.

Multiple instances don't support *vrf NAME*.

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

This sets the router-ID of the OSPF process. The router-ID may be an IP address of the router, but need not be - it can be any arbitrary 32bit number. However it MUST be unique within the entire OSPF domain to the OSPF speaker - bad things will happen if multiple OSPF speakers are configured with the same router-ID! If one is not specified then *ospfd* will obtain a router-ID automatically from *zebra*.

`ospf abr-type TYPE`

type can be cisco|ibm|shortcut|standard. The "Cisco" and "IBM" types are equivalent.

The OSPF standard for ABR behaviour does not allow an ABR to consider routes through non-backbone areas when its links to the backbone are down, even when there are other ABRs in attached non-backbone areas which still can reach the backbone - this restriction exists primarily to ensure routing-loops are avoided.

With the "Cisco" or "IBM" ABR type, the default in this release of FRR, this restriction is lifted, allowing an ABR to consider summaries learned from other ABRs through non-backbone areas, and hence route via non-backbone areas as a last resort when, and only when, backbone links are down.

Note that areas with fully-adjacent virtual-links are considered to be "transit capable" and can always be used to route backbone traffic, and hence are unaffected by this setting (*`area A.B.C.D virtual-link A.B.C.D`*).

More information regarding the behaviour controlled by this command can be found in **RFC 3509**, and *draft-ietf-ospf-shortcut-abr-02.txt*.

Quote: "Though the definition of the ABR in the OSPF specification does not require a router with multiple attached areas to have a backbone connection, it is actually necessary to provide successful routing to the inter-area and external destinations. If this requirement is not met, all traffic destined for the areas not connected to such an ABR or out of the OSPF domain, is dropped. This document describes alternative ABR behaviors implemented in Cisco and IBM routers."

`ospf rfc1583compatibility`

RFC 2328, the successor to **RFC 1583**, suggests according to section G.2 (changes) in section 16.4 a change to the path preference algorithm that prevents possible routing loops that were possible in the old version of OSPFv2. More specifically it demands that inter-area paths and intra-area backbone path are now of equal preference but still both preferred to external paths.

This command should NOT be set normally.

`log-adjacency-changes [detail]`

Configures ospfd to log changes in adjacency. With the optional detail argument, all changes in adjacency status are shown. Without detail, only changes to full or regressions are shown.

`passive-interface default`

Make all interfaces that belong to this router passive by default. For the description of passive interface look at *ip ospf passive [A.B.C.D]*. Per-interface configuration takes precedence over the default value.

`timers throttle spf (0-600000) (0-600000) (0-600000)`

This command sets the initial *delay*, the *initial-holdtime* and the *maximum-holdtime* between when SPF is calculated and the event which triggered the calculation. The times are specified in milliseconds and must be in the range of 0 to 600000 milliseconds.

The *delay* specifies the minimum amount of time to delay SPF calculation (hence it affects how long SPF calculation is delayed after an event which occurs outside of the holdtime of any previous SPF calculation, and also serves as a minimum holdtime).

Consecutive SPF calculations will always be separated by at least 'hold-time' milliseconds. The hold-time is adaptive and initially is set to the *initial-holdtime* configured with the above command. Events which occur within the holdtime of the previous SPF calculation will cause the holdtime to be increased by *initial-holdtime*, bounded by the *maximum-holdtime* configured with this command. If the adaptive hold-time elapses without any SPF-triggering event occurring then the current holdtime is reset to the *initial-holdtime*. The current holdtime can be viewed with `show ip ospf`, where it is expressed as a multiplier of the *initial-holdtime*.

```
router ospf
timers throttle spf 200 400 10000
```

In this example, the *delay* is set to 200ms, the initial holdtime is set to 400ms and the *maximum holdtime* to 10s. Hence there will always be at least 200ms between an event which requires SPF calculation and the actual SPF calculation. Further consecutive SPF calculations will always be separated by between 400ms to 10s, the hold-time increasing by 400ms each time an SPF-triggering event occurs within the hold-time of the previous SPF calculation.

This command supersedes the *timers spf* command in previous FRR releases.

`max-metric router-lsa [on-startup (5-86400)|on-shutdown (5-100)]`

`max-metric router-lsa administrative`

This enables **RFC 3137** support, where the OSPF process describes its transit links in its router-LSA as having infinite distance so that other routers will avoid calculating transit paths through the router while still being able to reach networks through the router.

This support may be enabled administratively (and indefinitely) or conditionally. Conditional enabling of max-metric router-lsas can be for a period of seconds after startup and/or for a period of seconds prior to shutdown.

Enabling this for a period after startup allows OSPF to converge fully first without affecting any existing routes used by other routers, while still allowing any connected stub links and/or redistributed routes to be reachable. Enabling this for a period of time in advance of shutdown allows the router to gracefully excuse itself from the OSPF domain.

Enabling this feature administratively allows for administrative intervention for whatever reason, for an indefinite period of time. Note that if the configuration is written to file, this administrative form of the stub-router command will also be written to file. If *ospfd* is restarted later, the command will then take effect until manually deconfigured.

Configured state of this feature as well as current status, such as the number of second remaining till on-startup or on-shutdown ends, can be viewed with the `show ip ospf` command.

`auto-cost reference-bandwidth (1-4294967)`

This sets the reference bandwidth for cost calculations, where this bandwidth is considered equivalent to an OSPF cost of 1, specified in Mbits/s. The default is 100Mbit/s (i.e. a link of bandwidth 100Mbit/s or higher will have a cost of 1. Cost of lower bandwidth links will be scaled with reference to this cost).

This configuration setting MUST be consistent across all routers within the OSPF domain.

`network A.B.C.D/M area A.B.C.D`

`network A.B.C.D/M area (0-4294967295)`

This command specifies the OSPF enabled interface(s). If the interface has an address from range 192.168.1.0/24 then the command below enables ospf on this interface so router can provide network information to the other ospf routers via this interface.

```
router ospf
network 192.168.1.0/24 area 0.0.0.0
```

Prefix length in interface must be equal or bigger (i.e. smaller network) than prefix length in network statement. For example statement above doesn't enable ospf on interface with address 192.168.1.1/23, but it does on interface with address 192.168.1.129/25.

Note that the behavior when there is a peer address defined on an interface changed after release 0.99.7. Currently, if a peer prefix has been configured, then we test whether the prefix in the network command contains the destination prefix. Otherwise, we test whether the network command prefix contains the local address prefix of the interface.

It is also possible to enable OSPF on a per interface/subnet basis using the interface command (*`ip ospf area AREA [ADDR]`*). However, mixing both network commands (`network`) and interface commands (`ip ospf`) on the same router is not supported.

`proactive-arp`

This command enables or disables sending ARP requests to update neighbor table entries. It speeds up convergence for /32 networks on a P2P connection.

This feature is enabled by default.

`clear ip ospf [(1-65535)] process`

This command can be used to clear the ospf process data structures. This will clear the ospf neighborship as well and it will get re-established. This will clear the LSDB too. This will be helpful when there is a change in router-id and if user wants the router-id change to take effect, user can use this cli instead of restarting the ospfd daemon.

`clear ip ospf [(1-65535)] neighbor`

This command can be used to clear the ospf neighbor data structures. This will clear the ospf neighborship and it will get re-established. This command can be used when the neighbor state get stuck at some state and this can be used to recover it from that state.

`maximum-paths (1-64)`

Use this command to control the maximum number of equal cost paths to reach a specific destination. The upper limit may differ if you change the value of MULTIPATH_NUM during compilation. The default is MULTIPATH_NUM (64).

`write-multiplier (1-100)`

Use this command to tune the amount of work done in the packet read and write threads before relinquishing control. The parameter is the number of packets to process before returning. The defult value of this parameter is 20.

`socket buffer <send | recv | all> (1-4000000000)`

This command controls the ospf instance's socket buffer sizes. The 'no' form resets one or both values to the default.

`no socket-per-interface`

Ordinarily, ospfd uses a socket per interface for sending packets. This command disables those per-interface sockets, and causes ospfd to use a single socket per ospf instance for sending and receiving packets.

Areas

`area A.B.C.D range A.B.C.D/M [advertise [cost (0-16777215)]]`

`area (0-4294967295) range A.B.C.D/M [advertise [cost (0-16777215)]]`

Summarize intra area paths from specified area into one Type-3 summary-LSA announced to other areas. This command can be used only in ABR and ONLY router-LSAs (Type-1) and network-LSAs (Type-2) (i.e. LSAs with scope area) can be summarized. Type-5 AS-external-LSAs can't be summarized - their scope is AS.

```
router ospf
 network 192.168.1.0/24 area 0.0.0.0
 network 10.0.0.0/8 area 0.0.0.10
 area 0.0.0.10 range 10.0.0.0/8
```

With configuration above one Type-3 Summary-LSA with routing info 10.0.0.0/8 is announced into backbone area if area 0.0.0.10 contains at least one intra-area network (i.e. described with router or network LSA) from this range.

`area A.B.C.D range A.B.C.D/M not-advertise`

`area (0-4294967295) range A.B.C.D/M not-advertise`

Instead of summarizing intra area paths filter them - i.e. intra area paths from this range are not advertised into other areas. This command makes sense in ABR only.

`area A.B.C.D range A.B.C.D/M {substitute A.B.C.D/M|cost (0-16777215)}`

`area (0-4294967295) range A.B.C.D/M {substitute A.B.C.D/M|cost (0-16777215)}`

Substitute summarized prefix with another prefix.

```
router ospf
 network 192.168.1.0/24 area 0.0.0.0
 network 10.0.0.0/8 area 0.0.0.10
 area 0.0.0.10 range 10.0.0.0/8 substitute 11.0.0.0/8
```

One Type-3 summary-LSA with routing info 11.0.0.0/8 is announced into backbone area if area 0.0.0.10 contains at least one intra-area network (i.e. described with router-LSA or network-LSA) from range 10.0.0.0/8.

By default, the metric of the summary route is calculated as the highest metric among the summarized routes. The *cost* option, however, can be used to set an explicit metric.

This command makes sense in ABR only.

`area A.B.C.D virtual-link A.B.C.D`

`area (0-4294967295) virtual-link A.B.C.D`

`area A.B.C.D shortcut`

`area (0-4294967295) shortcut`

Configure the area as Shortcut capable. See **RFC 3509**. This requires that the 'abr-type' be set to 'shortcut'.

`area A.B.C.D stub`

`area (0-4294967295) stub`

Configure the area to be a stub area. That is, an area where no router originates routes external to OSPF and hence an area where all external routes are via the ABR(s). Hence, ABRs for such an area do not need to pass AS-External LSAs (type-5s) or ASBR-Summary LSAs (type-4) into the area. They need only pass Network-Summary (type-3) LSAs into such an area, along with a default-route summary.

`area A.B.C.D stub no-summary`

`area (0-4294967295) stub no-summary`

Prevents an *ospfd* ABR from injecting inter-area summaries into the specified stub area.

`area A.B.C.D nssa`

`area (0-4294967295) nssa`

Configure the area to be a NSSA (Not-So-Stubby Area). This is an area that allows OSPF to import external routes into a stub area via a new LSA type (type 7). An NSSA autonomous system boundary router (ASBR) will generate this type of LSA. The area border router (ABR) translates the LSA type 7 into LSA type 5, which is propagated into the OSPF domain. NSSA areas are defined in RFC 3101.

`area A.B.C.D nssa suppress-fa`

`area (0-4294967295) nssa suppress-fa`

Configure the router to set the forwarding address to 0.0.0.0 in all LSA type 5 translated from LSA type 7. The router needs to be elected the translator of the area for this command to take effect. This feature causes routers that are configured not to advertise forwarding addresses into the backbone to direct forwarded traffic to the NSSA ABR translator.

`area A.B.C.D nssa default-information-originate [metric-type (1-2)] [metric (0-16777214)]`

`area (0-4294967295) nssa default-information-originate [metric-type (1-2)] [metric (0-16777214)]`

NSSA ABRs and ASBRs can be configured with the *default-information-originate* option to originate a Type-7 default route into the NSSA area. In the case of NSSA ASBRs, the origination of the default route is conditioned to the existence of a default route in the RIB that wasn't learned via the OSPF protocol.

`area A.B.C.D nssa range A.B.C.D/M [<not-advertise|cost (0-16777215)>]`

`area (0-4294967295) nssa range A.B.C.D/M [<not-advertise|cost (0-16777215)>]`

Summarize a group of external subnets into a single Type-7 LSA, which is then translated to a Type-5 LSA and avertised to the backbone. This command can only be used at the area boundary (NSSA ABR router).

By default, the metric of the summary route is calculated as the highest metric among the summarized routes. The *cost* option, however, can be used to set an explicit metric.

The *not-advertise* option, when present, prevents the summary route from being advertised, effectively filtering the summarized routes.

`area A.B.C.D default-cost (0-16777215)`

Set the cost of default-summary LSAs announced to stubby areas.

area A.B.C.D export-list NAME

area (0-4294967295) export-list NAME

Filter Type-3 summary-LSAs announced to other areas originated from intra- area paths from specified area.

```
router ospf
 network 192.168.1.0/24 area 0.0.0.0
 network 10.0.0.0/8 area 0.0.0.10
 area 0.0.0.10 export-list foo
!
access-list foo permit 10.10.0.0/16
access-list foo deny any
```

With example above any intra-area paths from area 0.0.0.10 and from range 10.10.0.0/16 (for example 10.10.1.0/24 and 10.10.2.128/30) are announced into other areas as Type-3 summary-LSA's, but any others (for example 10.11.0.0/16 or 10.128.30.16/30) aren't.

This command is only relevant if the router is an ABR for the specified area.

area A.B.C.D import-list NAME

area (0-4294967295) import-list NAME

Same as export-list, but it applies to paths announced into specified area as Type-3 summary-LSAs.

area A.B.C.D filter-list prefix NAME in

area A.B.C.D filter-list prefix NAME out

area (0-4294967295) filter-list prefix NAME in

area (0-4294967295) filter-list prefix NAME out

Filtering Type-3 summary-LSAs to/from area using prefix lists. This command makes sense in ABR only.

area A.B.C.D authentication

area (0-4294967295) authentication

Specify that simple password authentication should be used for the given area.

area A.B.C.D authentication message-digest

area (0-4294967295) authentication message-digest

Specify that OSPF packets must be authenticated with MD5 HMACs within the given area. Keying material must also be configured on a per-interface basis (`ip ospf message-digest-key`).

MD5 authentication may also be configured on a per-interface basis (*ip ospf authentication message-digest*). Such per-interface settings will override any per-area authentication setting.

Interfaces

ip ospf area AREA [ADDR]

Enable OSPF on the interface, optionally restricted to just the IP address given by *ADDR*, putting it in the *AREA* area. If you have a lot of interfaces, and/or a lot of subnets, then enabling OSPF via this command instead of (*network A.B.C.D/M area A.B.C.D*) may result in a slight performance improvement.

Notice that, mixing both network commands (`network`) and interface commands (`ip ospf`) on the same router is not supported. If (`ip ospf`) is present, (`network`) commands will fail.

ip ospf authentication-key AUTH_KEY

Set OSPF authentication key to a simple password. After setting *AUTH_KEY*, all OSPF packets are authenticated. *AUTH_KEY* has length up to 8 chars.

Simple text password authentication is insecure and deprecated in favour of MD5 HMAC authentication.

`ip ospf authentication message-digest`

Specify that MD5 HMAC authentication must be used on this interface. MD5 keying material must also be configured. Overrides any authentication enabled on a per-area basis (*`area A.B.C.D authentication message-digest`*)

Note that OSPF MD5 authentication requires that time never go backwards (correct time is NOT important, only that it never goes backwards), even across resets, if ospfd is to be able to promptly reestablish adjacencies with its neighbours after restarts/reboots. The host should have system time be set at boot from an external or non-volatile source (e.g. battery backed clock, NTP, etc.) or else the system clock should be periodically saved to non-volatile storage and restored at boot if MD5 authentication is to be expected to work reliably.

`ip ospf message-digest-key KEYID md5 KEY`

Set OSPF authentication key to a cryptographic password. The cryptographic algorithm is MD5.

KEYID identifies secret key used to create the message digest. This ID is part of the protocol and must be consistent across routers on a link.

KEY is the actual message digest key, of up to 16 chars (larger strings will be truncated), and is associated with the given KEYID.

`ip ospf cost (1-65535)`

Set link cost for the specified interface. The cost value is set to router-LSA's metric field and used for SPF calculation.

`ip ospf dead-interval (1-65535)`

`ip ospf dead-interval minimal hello-multiplier (2-20)`

Set number of seconds for RouterDeadInterval timer value used for Wait Timer and Inactivity Timer. This value must be the same for all routers attached to a common network. The default value is 40 seconds.

If 'minimal' is specified instead, then the dead-interval is set to 1 second and one must specify a hello-multiplier. The hello-multiplier specifies how many Hellos to send per second, from 2 (every 500ms) to 20 (every 50ms). Thus one can have 1s convergence time for OSPF. If this form is specified, then the hello-interval advertised in Hello packets is set to 0 and the hello-interval on received Hello packets is not checked, thus the hello-multiplier need NOT be the same across multiple routers on a common link.

`ip ospf hello-interval (1-65535)`

Set number of seconds for HelloInterval timer value. Setting this value, Hello packet will be sent every timer value seconds on the specified interface. This value must be the same for all routers attached to a common network. The default value is 10 seconds.

This command has no effect if *`ip ospf dead-interval minimal hello-multiplier (2-20)`* is also specified for the interface.

`ip ospf graceful-restart hello-delay (1-1800)`

Set the length of time during which Grace-LSAs are sent at 1-second intervals while coming back up after an unplanned outage. During this time, no hello packets are sent.

A higher hello delay will increase the chance that all neighbors are notified about the ongoing graceful restart before receiving a hello packet (which is crucial for the graceful restart to succeed). The hello delay shouldn't be set too high, however, otherwise the adjacencies might time out. As a best practice, it's recommended to set the hello delay and hello interval with the same values. The default value is 10 seconds.

`ip ospf network (broadcast|non-broadcast|point-to-multipoint [delay-reflood]|point-to-point [dmvpn])`

When configuring a point-to-point network on an interface and the interface has a /32 address associated with then OSPF will treat the interface as being *unnumbered*. If you are doing this you *must* set the net.ipv4.conf.<interface name>.rp_filter value to 0. In order for the ospf multicast packets to be delivered by the kernel.

When used in a DMVPN network at a spoke, this OSPF will be configured in point-to-point, but the HUB will be a point-to-multipoint. To make this topology work, specify the optional 'dmvpn' parameter at the spoke.

When the network is configured as point-to-multipoint and *delay-reflood* is specified, LSAs received on the interface from neighbors on the interface will not be flooded back out on the interface immediately. Rather, they will be added to the neighbor's link state retransmission list and only sent to the neighbor if the neighbor doesn't acknowledge the LSA prior to the link state retransmission timer expiring.

Set explicitly network type for specified interface.

`ip ospf priority (0-255)`

Set RouterPriority integer value. The router with the highest priority will be more eligible to become Designated Router. Setting the value to 0, makes the router ineligible to become Designated Router. The default value is 1.

`ip ospf retransmit-interval (1-65535)`

Set number of seconds for RxmtInterval timer value. This value is used when retransmitting Database Description and Link State Request packets. The default value is 5 seconds.

`ip ospf transmit-delay (1-65535) [A.B.C.D]`

Set number of seconds for InfTransDelay value. LSAs' age should be incremented by this value when transmitting. The default value is 1 second.

`ip ospf passive [A.B.C.D]`

Do not speak OSPF on the interface, but do advertise the interface as a stub link in the router-LSA for this router. This allows one to advertise addresses on such connected interfaces without having to originate AS-External/Type-5 LSAs (which have global flooding scope) - as would occur if connected addresses were redistributed into OSPF (*Redistribution*). This is the only way to advertise non-OSPF links into stub areas.

`ip ospf area (A.B.C.D|(0-4294967295))`

Enable ospf on an interface and set associated area.

3.11.3 OSPF route-map

Usage of *ospfd*'s route-map support.

`set metric [+|-](0-4294967295)`

Set a metric for matched route when sending announcement. Use plus (+) sign to add a metric value to an existing metric. Use minus (-) sign to substract a metric value from an existing metric.

Redistribution

`redistribute <babel|bgp|connected|eigrp|isis|kernel|openfabric|ospf|rip|sharp|static|table> [metric-typ`

Redistribute routes of the specified protocol or kind into OSPF, with the metric type and metric set if specified, filtering the routes using the given route-map if specified. Redistributed routes may also be filtered with distribute-lists, see *ospf distribute-list configuration*.

Redistributed routes are distributed as into OSPF as Type-5 External LSAs into links to areas that accept external routes, Type-7 External LSAs for NSSA areas and are not redistributed at all into Stub areas, where external routes are not permitted.

Note that for connected routes, one may instead use the `ip ospf passive [A.B.C.D]` configuration.

`default-information originate`

`default-information originate metric (0-16777214)`

`default-information originate metric (0-16777214) metric-type (1|2)`

`default-information originate metric (0-16777214) metric-type (1|2) route-map WORD`

default-information originate always

default-information originate always metric (0-16777214)

default-information originate always metric (0-16777214) metric-type (1|2)

default-information originate always metric (0-16777214) metric-type (1|2) route-map WORD

Originate an AS-External (type-5) LSA describing a default route into all external-routing capable areas, of the specified metric and metric type. If the 'always' keyword is given then the default is always advertised, even when there is no default present in the routing table.

distribute-list NAME out <kernel|connected|static|rip|isis|bgp|eigrp|nhrp|table|vnc|babel|openfabric>

Apply the access-list filter, NAME, to redistributed routes of the given type before allowing the routes to be redistributed into OSPF (*ospf redistribution*).

default-metric (0-16777214)

distance (1-255)

distance ospf (intra-area|inter-area|external) (1-255)

3.11.4 Graceful Restart

graceful-restart [grace-period (1-1800)]

Configure Graceful Restart (RFC 3623) restarting support. When enabled, the default grace period is 120 seconds.

To perform a graceful shutdown, the "graceful-restart prepare ip ospf" EXEC-level command needs to be issued before restarting the ospfd daemon.

When Graceful Restart is enabled and the ospfd daemon crashes or is killed abruptely (e.g. SIGKILL), it will attempt an unplanned Graceful Restart once it restarts.

graceful-restart helper enable [A.B.C.D]

Configure Graceful Restart (RFC 3623) helper support. By default, helper support is disabled for all neighbours. This config enables/disables helper support on this router for all neighbours. To enable/disable helper support for a specific neighbour, the router-id (A.B.C.D) has to be specified.

graceful-restart helper strict-lsa-checking

If 'strict-lsa-checking' is configured then the helper will abort the Graceful Restart when a LSA change occurs which affects the restarting router. By default 'strict-lsa-checking' is enabled"

graceful-restart helper supported-grace-time

Supports as HELPER for configured grace period.

graceful-restart helper planned-only

It helps to support as HELPER only for planned restarts. By default, it supports both planned and unplanned outages.

graceful-restart prepare ip ospf

Initiate a graceful restart for all OSPF instances configured with the "graceful-restart" command. The ospfd daemon should be restarted during the instance-specific grace period, otherwise the graceful restart will fail.

This is an EXEC-level command.

3.11.5 Showing Information

show ip ospf [vrf <NAME|all>] [json]
Show information on a variety of general OSPF and area state and configuration information.

show ip ospf interface [INTERFACE] [json]
Show state and configuration of OSPF the specified interface, or all interfaces if no interface is given.

show ip ospf neighbor [json]

show ip ospf [vrf <NAME|all>] neighbor INTERFACE [json]

show ip ospf neighbor detail [json]

show ip ospf [vrf <NAME|all>] neighbor A.B.C.D [detail] [json]

show ip ospf [vrf <NAME|all>] neighbor INTERFACE detail [json]
Display lsa information of LSDB. Json o/p of this command covers base route information i.e all LSAs except opaque lsa info.

show ip ospf [vrf <NAME|all>] database [self-originate] [json]
Show the OSPF database summary.

show ip ospf [vrf <NAME|all>] database max-age [json]
Show all MaxAge LSAs present in the OSPF link-state database.

show ip ospf [vrf <NAME|all>] database detail [LINK-STATE-ID] [adv-router A.B.C.D] [json]

show ip ospf [vrf <NAME|all>] database detail [LINK-STATE-ID] [self-originate] [json]

show ip ospf [vrf <NAME|all>] database (asbr-summary|external|network|router|summary|nssa-external|opaqu
B.C.D] [json]

show ip ospf [vrf <NAME|all>] database (asbr-summary|external|network|router|summary|nssa-external|opaqu
Show detailed information about the OSPF link-state database.

show ip ospf route [json]
Show the OSPF routing table, as determined by the most recent SPF calculation.

show ip ospf [vrf <NAME|all>] border-routers [json]
Show the list of ABR and ASBR border routers summary learnt via OSPFv2 Type-3 (Summary LSA) and Type-4 (Summary ASBR LSA). User can get that information as JSON format when `json` keyword at the end of cli is presented.

show ip ospf graceful-restart helper [detail] [json]
Displays the Grcaeful Restart Helper details including helper config changes.

3.11.6 Opaque LSA

ospf opaque-lsa

capability opaque
ospfd supports Opaque LSA (**RFC 2370**) as partial support for MPLS Traffic Engineering LSAs. The opaque-lsa capability must be enabled in the configuration. An alternate command could be "mpls-te on" (*Traffic Engineering*). Note that FRR offers only partial support for some of the routing protocol extensions that are used with MPLS-TE; it does not support a complete RSVP-TE solution.

show ip ospf [vrf <NAME|all>] database (opaque-link|opaque-area|opaque-external)

show ip ospf [vrf <NAME|all>] database (opaque-link|opaque-area|opaque-external) LINK-STATE-ID

show ip ospf [vrf <NAME|all>] database (opaque-link|opaque-area|opaque-external) LINK-STATE-ID adv-rout

show ip ospf [vrf <NAME|all>] database (opaque-link|opaque-area|opaque-external) adv-router ADV-ROUTER

show ip ospf [vrf <NAME|all>] database (opaque-link|opaque-area|opaque-external) LINK-STATE-ID self-ori

show ip ospf [vrf <NAME|all>] database (opaque-link|opaque-area|opaque-external) self-originate

Show Opaque LSA from the database.

show ip ospf (1-65535) reachable-routers

show ip ospf [vrf <NAME|all>] reachable-routers

Show routing table of reachable routers.

3.11.7 Traffic Engineering

Note: At this time, FRR offers partial support for some of the routing protocol extensions that can be used with MPLS-TE. FRR does not support a complete RSVP-TE solution currently.

mpls-te on

Enable Traffic Engineering LSA flooding.

mpls-te router-address <A.B.C.D>

Configure stable IP address for MPLS-TE. This IP address is then advertise in Opaque LSA Type-10 TLV=1 (TE) option 1 (Router-Address).

mpls-te inter-as area <area-id>|as

Enable RFC 5392 support - Inter-AS TE v2 - to flood Traffic Engineering parameters of Inter-AS link. 2 modes are supported: AREA and AS; LSA are flood in AREA <area-id> with Opaque Type-10, respectively in AS with Opaque Type-11. In all case, Opaque-LSA TLV=6.

mpls-te export

Export Traffic Engineering Data Base to other daemons through the ZAPI Opaque Link State messages.

show ip ospf mpls-te interface

show ip ospf mpls-te interface INTERFACE

Show MPLS Traffic Engineering parameters for all or specified interface.

show ip ospf mpls-te router

Show Traffic Engineering router parameters.

show ip ospf mpls-te database [verbose|json]

show ip ospf mpls-te database vertex [self-originate|adv-router ADV-ROUTER] [verbose|json]

show ip ospf mpls-te database edge [A.B.C.D] [verbose|json]

show ip ospf mpls-te database subnet [A.B.C.D/M] [verbose|json]

Show Traffic Engineering Database

3.11.8 Router Information

`router-info [as | area]`
: Enable Router Information (**RFC 4970**) LSA advertisement with AS scope (default) or Area scope flooding when area is specified. Old syntax *router-info area <A.B.C.D>* is always supported but mark as deprecated as the area ID is no more necessary. Indeed, router information support multi-area and detect automatically the areas.

`pce address <A.B.C.D>`

`pce domain as (0-65535)`

`pce neighbor as (0-65535)`

`pce flag BITPATTERN`

`pce scope BITPATTERN`
: The commands are conform to **RFC 5088** and allow OSPF router announce Path Computation Element (PCE) capabilities through the Router Information (RI) LSA. Router Information must be enable prior to this. The command set/unset respectively the PCE IP address, Autonomous System (AS) numbers of controlled domains, neighbor ASs, flag and scope. For flag and scope, please refer to :rfc`5088` for the BITPATTERN recognition. Multiple 'pce neighbor' command could be specified in order to specify all PCE neighbours.

`show ip ospf router-info`
: Show Router Capabilities flag.

`show ip ospf router-info pce`
: Show Router Capabilities PCE parameters.

3.11.9 Segment Routing

This is an EXPERIMENTAL support of Segment Routing as per *RFC 8665* for MPLS dataplane.

`segment-routing on`
: Enable Segment Routing. Even if this also activate routing information support, it is preferable to also activate routing information, and set accordingly the Area or AS flooding.

`segment-routing global-block (16-1048575) (16-1048575) [local-block (16-1048575) (16-1048575)]`
: Set the Segment Routing Global Block i.e. the label range used by MPLS to store label in the MPLS FIB for Prefix SID. Optionally also set the Local Block, i.e. the label range used for Adjacency SID. The negative version of the command always unsets both ranges.

`segment-routing node-msd (1-16)`
: Fix the Maximum Stack Depth supported by the router. The value depend of the MPLS dataplane. E.g. for Linux kernel, since version 4.13 it is 32.

`segment-routing prefix A.B.C.D/M [index (0-65535)|no-php-flag|explicit-null]`
: prefix with /32 corresponding to a loopback interface are currently supported. The 'no-php-flag' means NO Penultimate Hop Popping that allows SR node to request to its neighbor to not pop the label. The 'explicit-null' means that neighbor nodes must swap the incoming label by the MPLS Explicit Null label before delivering the packet.

`show ip ospf database segment-routing <adv-router ADVROUTER|self-originate> [json]`
: Show Segment Routing Data Base, all SR nodes, specific advertised router or self router. Optional JSON output can be obtained by appending 'json' to the end of the command.

3.11.10 External Route Summarisation

This feature summarises originated external LSAs(Type-5 and Type-7). Summary Route will be originated on-behalf of all matched external LSAs.

`summary-address A.B.C.D/M [tag (1-4294967295)]`
: This command enable/disables summarisation for the configured address range. Tag is the optional parameter. If tag configured Summary route will be originated with the configured tag.

`summary-address A.B.C.D/M no-advertise`
: This command to ensure not advertise the summary lsa for the matched external LSAs.

`aggregation timer (5-1800)`
: Configure aggregation delay timer interval. Summarisation starts only after this delay timer expiry. By default, delay interval is 5 seconds.

  The no form of the command resets the aggregation delay interval to default value.

`show ip ospf [vrf <NAME|all>] summary-address [detail] [json]`
: Show configuration for display all configured summary routes with matching external LSA information.

3.11.11 TI-LFA

Experimental support for Topology Independent LFA (Loop-Free Alternate), see for example 'draft-bashandy-rtgwg-segment-routing-ti-lfa-05'. Note that TI-LFA requires a proper Segment Routing configuration.

`fast-reroute ti-lfa [node-protection]`
: Configured on the router level. Activates TI-LFA for all interfaces.

  Note that so far only P2P interfaces are supported.

3.11.12 Debugging OSPF

`debug ospf [(1-65535)] bfd`
: Enable or disable debugging for BFD events. This will show BFD integration library messages and OSPF BFD integration messages that are mostly state transitions and validation problems.

`debug ospf [(1-65535)] client-api`
: Show debug information for the OSPF opaque data client API.

`debug ospf [(1-65535)] default-information`
: Show debug information of default information

`debug ospf [(1-65535)] packet (hello|dd|ls-request|ls-update|ls-ack|all) (send|recv) [detail]`
: Dump Packet for debugging

`debug ospf [(1-65535)] ism [status|events|timers]`
: Show debug information of Interface State Machine

`debug ospf [(1-65535)] nsm [status|events|timers]`
: Show debug information of Network State Machine

`debug ospf [(1-65535)] event`
: Show debug information of OSPF event

`debug ospf [(1-65535)] nssa`
: Show debug information about Not So Stub Area

`debug ospf [(1-65535)] ldp-sync`
: Show debug information about LDP-Sync

debug ospf [(1-65535)] lsa [aggregate|flooding|generate|install|refresh]
Show debug detail of Link State messages

debug ospf [(1-65535)] sr
Show debug information about Segment Routing

debug ospf [(1-65535)] te
Show debug information about Traffic Engineering LSA

debug ospf [(1-65535)] ti-lfa
Show debug information about SR TI-LFA

debug ospf [(1-65535)] zebra [interface|redistribute]
Show debug information of ZEBRA API

debug ospf [(1-65535)] graceful-restart
Enable/disable debug information for OSPF Graceful Restart Helper

show debugging ospf

3.11.13 Sample Configuration

A simple example, with MD5 authentication enabled:

```
!
interface bge0
 ip ospf authentication message-digest
 ip ospf message-digest-key 1 md5 ABCDEFGHIJK
!
router ospf
 network 192.168.0.0/16 area 0.0.0.1
 area 0.0.0.1 authentication message-digest
```

An ABR router, with MD5 authentication and performing summarisation of networks between the areas:

```
!
password ABCDEF
log file /var/log/frr/ospfd.log
service advanced-vty
!
interface eth0
 ip ospf authentication message-digest
 ip ospf message-digest-key 1 md5 ABCDEFGHIJK
!
interface ppp0
 ip ospf passive
!
interface br0
 ip ospf authentication message-digest
 ip ospf message-digest-key 2 md5 XYZ12345
!
router ospf
 ospf router-id 192.168.0.1
 redistribute connected
 network 192.168.0.0/24 area 0.0.0.0
 network 10.0.0.0/16 area 0.0.0.0
```

(continues on next page)

(continued from previous page)

```
 network 192.168.1.0/24 area 0.0.0.1
 area 0.0.0.0 authentication message-digest
 area 0.0.0.0 range 10.0.0.0/16
 area 0.0.0.0 range 192.168.0.0/24
 area 0.0.0.1 authentication message-digest
 area 0.0.0.1 range 10.2.0.0/16
!
```

A Traffic Engineering configuration, with Inter-ASv2 support.

First, the `zebra.conf` part:

```
interface eth0
 ip address 198.168.1.1/24
 link-params
  enable
  admin-grp 0xa1
  metric 100
  max-bw 1.25e+07
  max-rsv-bw 1.25e+06
  unrsv-bw 0 1.25e+06
  unrsv-bw 1 1.25e+06
  unrsv-bw 2 1.25e+06
  unrsv-bw 3 1.25e+06
  unrsv-bw 4 1.25e+06
  unrsv-bw 5 1.25e+06
  unrsv-bw 6 1.25e+06
  unrsv-bw 7 1.25e+06
!
interface eth1
 ip address 192.168.2.1/24
 link-params
  enable
  metric 10
  max-bw 1.25e+07
  max-rsv-bw 1.25e+06
  unrsv-bw 0 1.25e+06
  unrsv-bw 1 1.25e+06
  unrsv-bw 2 1.25e+06
  unrsv-bw 3 1.25e+06
  unrsv-bw 4 1.25e+06
  unrsv-bw 5 1.25e+06
  unrsv-bw 6 1.25e+06
  unrsv-bw 7 1.25e+06
  neighbor 192.168.2.2 as 65000
   hostname HOSTNAME
   password PASSWORD
   log file /var/log/zebra.log
   !
   interface eth0
    ip address 198.168.1.1/24
    link-params
     enable
```

(continues on next page)

(continued from previous page)

```
     admin-grp 0xa1
     metric 100
     max-bw 1.25e+07
     max-rsv-bw 1.25e+06
     unrsv-bw 0 1.25e+06
     unrsv-bw 1 1.25e+06
     unrsv-bw 2 1.25e+06
     unrsv-bw 3 1.25e+06
     unrsv-bw 4 1.25e+06
     unrsv-bw 5 1.25e+06
     unrsv-bw 6 1.25e+06
     unrsv-bw 7 1.25e+06
   !
   interface eth1
    ip address 192.168.2.1/24
    link-params
     enable
     metric 10
     max-bw 1.25e+07
     max-rsv-bw 1.25e+06
     unrsv-bw 0 1.25e+06
     unrsv-bw 1 1.25e+06
     unrsv-bw 2 1.25e+06
     unrsv-bw 3 1.25e+06
     unrsv-bw 4 1.25e+06
     unrsv-bw 5 1.25e+06
     unrsv-bw 6 1.25e+06
     unrsv-bw 7 1.25e+06
     neighbor 192.168.2.2 as 65000
```

Then the ospfd.conf itself:

```
hostname HOSTNAME
password PASSWORD
log file /var/log/ospfd.log
!
!
interface eth0
 ip ospf hello-interval 60
 ip ospf dead-interval 240
!
interface eth1
 ip ospf hello-interval 60
 ip ospf dead-interval 240
!
!
router ospf
 ospf router-id 192.168.1.1
 network 192.168.0.0/16 area 1
 ospf opaque-lsa
 mpls-te
 mpls-te router-address 192.168.1.1
```

(continues on next page)

(continued from previous page)

```
 mpls-te inter-as area 1
!
line vty
```

A router information example with PCE advertisement:

```
!
router ospf
 ospf router-id 192.168.1.1
 network 192.168.0.0/16 area 1
 capability opaque
 mpls-te
 mpls-te router-address 192.168.1.1
 router-info area 0.0.0.1
 pce address 192.168.1.1
 pce flag 0x80
 pce domain as 65400
 pce neighbor as 65500
 pce neighbor as 65200
 pce scope 0x80
!
```

3.12 OSPFv3

ospf6d is a daemon support OSPF version 3 for IPv6 network. OSPF for IPv6 is described in **RFC 2740**.

3.12.1 OSPF6 router

`router ospf6 [vrf NAME]`

`ospf6 router-id A.B.C.D`

Set router's Router-ID.

`timers throttle spf (0-600000) (0-600000) (0-600000)`

This command sets the initial *delay*, the *initial-holdtime* and the *maximum-holdtime* between when SPF is calculated and the event which triggered the calculation. The times are specified in milliseconds and must be in the range of 0 to 600000 milliseconds.

The *delay* specifies the minimum amount of time to delay SPF calculation (hence it affects how long SPF calculation is delayed after an event which occurs outside of the holdtime of any previous SPF calculation, and also serves as a minimum holdtime).

Consecutive SPF calculations will always be separated by at least 'hold-time' milliseconds. The hold-time is adaptive and initially is set to the *initial-holdtime* configured with the above command. Events which occur within the holdtime of the previous SPF calculation will cause the holdtime to be increased by *initial-holdtime*, bounded by the *maximum-holdtime* configured with this command. If the adaptive hold-time elapses without any SPF-triggering event occurring then the current holdtime is reset to the *initial-holdtime*.

```
router ospf6
 timers throttle spf 200 400 10000
```

In this example, the *delay* is set to 200ms, the initial holdtime is set to 400ms and the *maximum holdtime* to 10s. Hence there will always be at least 200ms between an event which requires SPF calculation and the actual SPF calculation. Further consecutive SPF calculations will always be separated by between 400ms to 10s, the hold-time increasing by 400ms each time an SPF-triggering event occurs within the hold-time of the previous SPF calculation.

`auto-cost reference-bandwidth COST`

This sets the reference bandwidth for cost calculations, where this bandwidth is considered equivalent to an OSPF cost of 1, specified in Mbits/s. The default is 100Mbit/s (i.e. a link of bandwidth 100Mbit/s or higher will have a cost of 1. Cost of lower bandwidth links will be scaled with reference to this cost).

This configuration setting MUST be consistent across all routers within the OSPF domain.

`maximum-paths (1-64)`

Use this command to control the maximum number of parallel routes that OSPFv3 can support. The default is 64.

`write-multiplier (1-100)`

Use this command to tune the amount of work done in the packet read and write threads before relinquishing control. The parameter is the number of packets to process before returning. The default value of this parameter is 20.

`clear ipv6 ospf6 process [vrf NAME]`

This command clears up the database and routing tables and resets the neighborship by restarting the interface state machine. This will be helpful when there is a change in router-id and if user wants the router-id change to take effect, user can use this cli instead of restarting the ospf6d daemon.

`clear ipv6 ospf6 [vrf NAME] interface [IFNAME]`

This command restarts the interface state machine for all interfaces in the VRF or only for the specific interface if `IFNAME` is specified.

3.12.2 ASBR Summarisation Support in OSPFv3

External routes in OSPFv3 are carried by type 5/7 LSA (external LSAs). External LSAs are generated by ASBR (Autonomous System Boundary Router). Large topology database requires a large amount of router memory, which slows down all processes, including SPF calculations. It is necessary to reduce the size of the OSPFv3 topology database, especially in a large network. Summarising routes keeps the routing tables smaller and easier to troubleshoot.

External route summarization must be configured on ASBR. Stub area do not allow ASBR because they don't allow type 5 LSAs.

An ASBR will inject a summary route into the OSPFv3 domain.

Summary route will only be advertised if you have at least one subnet that falls within the summary range.

Users will be allowed an option in the CLI to not advertise range of ipv6 prefixes as well.

The configuration of ASBR Summarisation is supported using the CLI command

`summary-address X:X::X:X/`
`M [tag (1-4294967295)] [{metric (0-16777215) | metric-type (1-2)}]`

This command will advertise a single External LSA on behalf of all the prefixes falling under this range configured by the CLI. The user is allowed to configure tag, metric and metric-type as well. By default, tag is not configured, default metric as 20 and metric-type as type-2 gets advertised. A summary route is created when one or more specific routes are learned and removed when no more specific route exist. The summary route is also installed in the local system with Null0 as next-hop to avoid leaking traffic.

`no summary-address X:X::X:X/`
`M [tag (1-4294967295)] [{metric (0-16777215) | metric-type (1-2)}]`

This command can be used to remove the summarisation configuration. This will flush the single External LSA if it was originated and advertise the External LSAs for all the existing individual prefixes.

`summary-address X:X::X:X/M no-advertise`

This command can be used when user do not want to advertise a certain range of prefixes using the no-advertise option. This command when configured will flush all the existing external LSAs falling under this range.

`no summary-address X:X::X:X/M no-advertise`

This command can be used to remove the previous configuration. When configured, tt will resume originating external LSAs for all the prefixes falling under the configured range.

`aggregation timer (5-1800)`

The summarisation command takes effect after the aggregation timer expires. By default the value of this timer is 5 seconds. User can modify the time after which the external LSAs should get originated using this command.

`no aggregation timer (5-1800)`

This command removes the timer configuration. It reverts back to default 5 second timer.

`show ipv6 ospf6 summary-address [detail] [json]`

This command can be used to see all the summary-address related information. When detail option is used, it shows all the prefixes falling under each summary-configuration apart from other information.

3.12.3 OSPF6 area

`area A.B.C.D range X:X::X:X/M [<advertise|not-advertise|cost (0-16777215)>]`

`area (0-4294967295) range X:X::X:X/M [<advertise|not-advertise|cost (0-16777215)>]`

Summarize a group of internal subnets into a single Inter-Area-Prefix LSA. This command can only be used at the area boundary (ABR router).

By default, the metric of the summary route is calculated as the highest metric among the summarized routes. The *cost* option, however, can be used to set an explicit metric.

The *not-advertise* option, when present, prevents the summary route from being advertised, effectively filtering the summarized routes.

`area A.B.C.`
`D nssa [no-summary] [default-information-originate [metric-type (1-2)] [metric (0-16777214)]]`

`area (0-4294967295) nssa [no-summary] [default-information-originate [metric-type (1-2)] [metric (0-167`

Configure the area to be a NSSA (Not-So-Stubby Area).

The following functionalities are implemented as per RFC 3101:

1. Advertising Type-7 LSA into NSSA area when external route is redistributed into OSPFv3.
2. Processing Type-7 LSA received from neighbor and installing route in the route table.
3. Support for NSSA ABR functionality which is generating Type-5 LSA when backbone area is configured. Currently translation of Type-7 LSA to Type-5 LSA is enabled by default.
4. Support for NSSA Translator functionality when there are multiple NSSA ABR in an area.

An NSSA ABR can be configured with the *no-summary* option to prevent the advertisement of summaries into the area. In that case, a single Type-3 LSA containing a default route is originated into the NSSA.

NSSA ABRs and ASBRs can be configured with *default-information-originate* option to originate a Type-7 default route into the NSSA area. In the case of NSSA ASBRs, the origination of the default route is conditioned to the existence of a default route in the RIB that wasn't learned via the OSPF protocol.

area A.B.C.D nssa range X:X::X:X/M [<not-advertise|cost (0-16777215)>]

area (0-4294967295) nssa range X:X::X:X/M [<not-advertise|cost (0-16777215)>]

Summarize a group of external subnets into a single Type-7 LSA, which is then translated to a Type-5 LSA and avertised to the backbone. This command can only be used at the area boundary (NSSA ABR router).

By default, the metric of the summary route is calculated as the highest metric among the summarized routes. The *cost* option, however, can be used to set an explicit metric.

The *not-advertise* option, when present, prevents the summary route from being advertised, effectively filtering the summarized routes.

area A.B.C.D export-list NAME

area (0-4294967295) export-list NAME

Filter Type-3 summary-LSAs announced to other areas originated from intra- area paths from specified area.

```
router ospf6
 area 0.0.0.10 export-list foo
!
ipv6 access-list foo permit 2001:db8:1000::/64
ipv6 access-list foo deny any
```

With example above any intra-area paths from area 0.0.0.10 and from range 2001:db8::/32 (for example 2001:db8:1::/64 and 2001:db8:2::/64) are announced into other areas as Type-3 summary-LSA's, but any others (for example 2001:200::/48) aren't.

This command is only relevant if the router is an ABR for the specified area.

area A.B.C.D import-list NAME

area (0-4294967295) import-list NAME

Same as export-list, but it applies to paths announced into specified area as Type-3 summary-LSAs.

area A.B.C.D filter-list prefix NAME in

area A.B.C.D filter-list prefix NAME out

area (0-4294967295) filter-list prefix NAME in

area (0-4294967295) filter-list prefix NAME out

Filtering Type-3 summary-LSAs to/from area using prefix lists. This command makes sense in ABR only.

3.12.4 OSPF6 interface

ipv6 ospf6 area <A.B.C.D|(0-4294967295)>

Enable OSPFv3 on the interface and add it to the specified area.

ipv6 ospf6 cost COST

Sets interface's output cost. Default value depends on the interface bandwidth and on the auto-cost reference bandwidth.

ipv6 ospf6 hello-interval HELLOINTERVAL

Sets interface's Hello Interval. Default 10

ipv6 ospf6 dead-interval DEADINTERVAL

Sets interface's Router Dead Interval. Default value is 40.

ipv6 ospf6 graceful-restart hello-delay HELLODELAYINTERVAL

Set the length of time during which Grace-LSAs are sent at 1-second intervals while coming back up after an unplanned outage. During this time, no hello packets are sent.

A higher hello delay will increase the chance that all neighbors are notified about the ongoing graceful restart before receiving a hello packet (which is crucial for the graceful restart to succeed). The hello delay shouldn't be set too high, however, otherwise the adjacencies might time out. As a best practice, it's recommended to set the hello delay and hello interval with the same values. The default value is 10 seconds.

`ipv6 ospf6 retransmit-interval RETRANSMITINTERVAL`
Sets interface's Rxmt Interval. Default value is 5.

`ipv6 ospf6 priority PRIORITY`
Sets interface's Router Priority. Default value is 1.

`ipv6 ospf6 transmit-delay TRANSMITDELAY`
Sets interface's Inf-Trans-Delay. Default value is 1.

`ipv6 ospf6 network (broadcast|point-to-point)`
Set explicitly network type for specified interface.

3.12.5 OSPF6 route-map

Usage of *ospfd6*'s route-map support.

`set metric [+|-](0-4294967295)`
Set a metric for matched route when sending announcement. Use plus (+) sign to add a metric value to an existing metric. Use minus (-) sign to substract a metric value from an existing metric.

3.12.6 Redistribute routes to OSPF6

`redistribute <babel|bgp|connected|isis|kernel|openfabric|ripng|sharp|static|table> [metric-type (1-2)]`
Redistribute routes of the specified protocol or kind into OSPFv3, with the metric type and metric set if specified, filtering the routes using the given route-map if specified.

`default-information originate [{always|metric (0-16777214)|metric-type (1-2)|route-map WORD}]`
The command injects default route in the connected areas. The always argument injects the default route regardless of it being present in the router. Metric values and route-map can also be specified optionally.

3.12.7 Graceful Restart

`graceful-restart [grace-period (1-1800)]`
Configure Graceful Restart (RFC 5187) restarting support. When enabled, the default grace period is 120 seconds.

To perform a graceful shutdown, the "graceful-restart prepare ipv6 ospf" EXEC-level command needs to be issued before restarting the ospf6d daemon.

When Graceful Restart is enabled and the ospf6d daemon crashes or is killed abruptely (e.g. SIGKILL), it will attempt an unplanned Graceful Restart once it restarts.

`graceful-restart helper enable [A.B.C.D]`
Configure Graceful Restart (RFC 5187) helper support. By default, helper support is disabled for all neighbours. This config enables/disables helper support on this router for all neighbours. To enable/disable helper support for a specific neighbour, the router-id (A.B.C.D) has to be specified.

`graceful-restart helper strict-lsa-checking`
If 'strict-lsa-checking' is configured then the helper will abort the Graceful Restart when a LSA change occurs which affects the restarting router. By default 'strict-lsa-checking' is enabled"

`graceful-restart helper supported-grace-time (10-1800)`

Supports as HELPER for configured grace period.

`graceful-restart helper planned-only`

It helps to support as HELPER only for planned restarts. By default, it supports both planned and unplanned outages.

`graceful-restart prepare ipv6 ospf`

Initiate a graceful restart for all OSPFv3 instances configured with the "graceful-restart" command. The ospf6d daemon should be restarted during the instance-specific grace period, otherwise the graceful restart will fail.

This is an EXEC-level command.

3.12.8 Authentication trailer support:

IPv4 version of OSPF supports authentication as part of the base RFC. When IPv6 version of OSPF was developed there was IPSec support for IPv6, Hence OSPFv3(IPv6 version of OSPF) suggest to use IPSec as authentication and encryption mechanism. IPSec supports authentication using AH header and Encryption using ESP.

There are few disadvantages of using IPSec with OSPFv3.

1. If encryption is enabled for OSPFv3 packets, then its not possible to give priority to control packets.
2. IPSec has platform dependency and may not be supported in all platforms.
3. It is performance intensive.
4. Its difficult to configure.

Some advantages of OSPFv3 authentication trailer feature.

1. It provides replay protection via sequence number.
2. It provides IPv6 source address protection.
3. No platform dependency.
4. Easy to implement and maintain.

This feature is support for `RFC7166`.

FRR supports MD5 and SHA256 internally and relays on openssl for other hash algorithms. If user wants to use only MD5 and SHA256, no special action is required. If user wants complete support of authentication trailer with all hash algorithms follow below steps.

Installing Dependencies:

```
sudo apt update
sudo apt-get install openssl
```

Compile:

Follow normal compilation as mentioned in the build page. If you want to use all the hash algorithms then follow the steps mentioned in note before compiling.

Note: If your platform supports `openssl`, please make sure to add `--with-crypto=openssl` to your configure options. Default value is `--with-crypto=internal`

CLI Configuration:

There are two ways in which authentication trailer can be configured for OSPFv3. These commands are mutually exclusive, only one can be configured at any time.

1. Using manual key configuration.
2. Using keychain.

List of hash algorithms supported:

Without openssl:

`MD5` `HMAC-SHA-256`

With openssl:

`MD5` `HMAC-SHA-1` `HMAC-SHA-256` `HMAC-SHA-384` `HMAC-SHA-512`

Example configuration of manual key:

Without openssl:

```
ipv6 ospf6 authentication key-id (1-65535) hash-algo <md5|hmac-sha-256> key WORD
```

With openssl:

```
ipv6 ospf6 authentication key-id (1-65535) hash-algo <md5|hmac-sha-256|hmac-sha-1|hmac-sha-384|hmac-sha-
```

Example configuration of keychain:

```
ipv6 ospf6 authentication keychain KEYCHAIN_NAME
```

Running configuration:

Manual key:

```
frr# show running-config
Building configuration...

Current configuration:
!
interface ens192
 ipv6 address 2001:DB8::2/64
 ipv6 ospf6 authentication key-id 10 hash-algo hmac-sha-256 key abhinay
```

Keychain:

```
frr# show running-config
Building configuration...

Current configuration:
!
interface ens192
 ipv6 address 2001:DB8::2/64
 ipv6 ospf6 authentication keychain abhinay
```

Example keychain config:

```
frr#show running-config
Building configuration...

Current configuration:
!
 key chain abcd
  key 100
   key-string password
   cryptographic-algorithm sha1
  exit
  key 200
   key-string password
   cryptographic-algorithm sha256
  exit
 !
 key chain pqr
  key 300
   key-string password
   cryptographic-algorithm sha384
  exit
  key 400
   key-string password
   cryptographic-algorithm sha384
```

(continues on next page)

(continued from previous page)

```
  exit
 !
```

Show commands:

There is an interface show command that displays if authentication trailer is enabled or not. json output is also supported.

There is support for drop counters, which will help in debugging the feature.

```
frr# show ipv6 ospf6 interface ens192
ens192 is up, type BROADCAST
  Interface ID: 5
  Number of I/F scoped LSAs is 2
    0 Pending LSAs for LSUpdate in Time 00:00:00 [thread off]
    0 Pending LSAs for LSAck in Time 00:00:00 [thread off]
  Authentication trailer is enabled with manual key         ==> new info added
    Packet drop Tx 0, Packet drop Rx 0
```

OSPFv3 supports options in hello and database description packets hence the presence of authentication trailer needs to be stored in OSPFv3 neighbor info. Since RFC specifies that we need to handled sequence number for every ospf6 packet type, sequence number recvd in authentication header from the neighbor is stored in neighbor to validate the packet. json output is also supported.

```
frr# show ipv6 ospf6 neighbor 2.2.2.2 detail
 Neighbor 2.2.2.2%ens192
    Area 1 via interface ens192 (ifindex 3)
    0 Pending LSAs for LSUpdate in Time 00:00:00 [thread off]
    0 Pending LSAs for LSAck in Time 00:00:00 [thread off]
    Authentication header present                           ==> new info added
                         hello         DBDesc        LSReq         LSUpd         LSAck
      Higher sequence no 0x0           0x0           0x0           0x0           0x0
      Lower sequence no  0x242E        0x1DC4        0x1DC3        0x23CC        0x1DDA
```

Sent packet sequence number is maintained per ospf6 router for every packet that is sent out of router, so sequence number is maintained per ospf6 process.

```
frr# show ipv6 ospf6
 OSPFv3 Routing Process (0) with Router-ID 2.2.2.2
 Number of areas in this router is 1
 Authentication Sequence number info
  Higher sequence no 3, Lower sequence no 1656
```

Debug command:

Below command can be used to enable ospfv3 authentication trailer specific logs if you have to debug the feature.

debug ospf6 authentication [<tx|rx>]

Feature supports authentication trailer tx/rx drop counters for debugging, which can be used to see if packets are getting dropped due to error in processing authentication trailer information in OSPFv3 packet. json output is also supported.

```
frr# show ipv6 ospf6 interface ens192
ens192 is up, type BROADCAST
  Interface ID: 5
  Number of I/F scoped LSAs is 2
    0 Pending LSAs for LSUpdate in Time 00:00:00 [thread off]
    0 Pending LSAs for LSAck in Time 00:00:00 [thread off]
  Authentication trailer is enabled with manual key
    Packet drop Tx 0, Packet drop Rx 0                      ==> new counters
```

Clear command:

Below command can be used to clear the tx/rx drop counters in interface. Below command can be used to clear all ospfv3 interface or specific interface by specifying the interface name.

clear ipv6 ospf6 auth-counters interface [IFNAME]

3.12.9 Showing OSPF6 information

show ipv6 ospf6 [vrf <NAME|all>] [json]
Show information on a variety of general OSPFv3 and area state and configuration information. JSON output can be obtained by appending 'json' to the end of command.

show ipv6 ospf6 [vrf <NAME|all>] database [<detail|dump|internal>] [json]
This command shows LSAs present in the LSDB. There are three view options. These options helps in viewing all the parameters of the LSAs. JSON output can be obtained by appending 'json' to the end of command. JSON option is not applicable with 'dump' option.

show ipv6 ospf6 [vrf <NAME|all>] database <router|network|inter-prefix|inter-router|as-external|group-m
These options filters out the LSA based on its type. The three views options works here as well. JSON output can be obtained by appending 'json' to the end of command.

show ipv6 ospf6 [vrf <NAME|all>] database adv-router A.B.C.D linkstate-id A.B.C.D [json]
The LSAs additinally can also be filtered with the linkstate-id and advertising-router fields. We can use the LSA type filter and views with this command as well and visa-versa. JSON output can be obtained by appending 'json' to the end of command.

show ipv6 ospf6 [vrf <NAME|all>] database self-originated [json]
This command is used to filter the LSAs which are originated by the present router. All the other filters are applicable here as well.

show ipv6 ospf6 [vrf <NAME|all>] interface [json]
To see OSPF interface configuration like costs. JSON output can be obtained by appending "json" in the end.

show ipv6 ospf6 [vrf <NAME|all>] neighbor [json]
Shows state and chosen (Backup) DR of neighbor. JSON output can be obtained by appending 'json' at the end.

`show ipv6 ospf6 [vrf <NAME|all>] interface traffic [json]`
: Shows counts of different packets that have been received and transmitted by the interfaces. JSON output can be obtained by appending "json" at the end.

`show ipv6 route ospf6`
: This command shows internal routing table.

`show ipv6 ospf6 zebra [json]`
: Shows state about what is being redistributed between zebra and OSPF6. JSON output can be obtained by appending "json" at the end.

`show ipv6 ospf6 [vrf <NAME|all>] redistribute [json]`
: Shows the routes which are redistributed by the router. JSON output can be obtained by appending 'json' at the end.

`show ipv6 ospf6 [vrf <NAME|all>] route [<intra-area|inter-area|external-1|external-2|X:X::X:X|X:X::X:X/M|detail|summary>] [json]`
: This command displays the ospfv3 routing table as determined by the most recent SPF calculations. Options are provided to view the different types of routes. Other than the standard view there are two other options, detail and summary. JSON output can be obtained by appending 'json' to the end of command.

`show ipv6 ospf6 [vrf <NAME|all>] route X:X::X:X/M match [detail] [json]`
: The additional match option will match the given address to the destination of the routes, and return the result accordingly.

`show ipv6 ospf6 [vrf <NAME|all>] interface [IFNAME] prefix [detail|<X:X::X:X|X:X::X:X/M> [<match|detail>]] [json]`
: This command shows the prefixes present in the interface routing table. Interface name can also be given. JSON output can be obtained by appending 'json' to the end of command.

`show ipv6 ospf6 [vrf <NAME|all>] spf tree [json]`
: This commands shows the spf tree from the recent spf calculation with the calling router as the root. If json is appended in the end, we can get the tree in JSON format. Each area that the router belongs to has it's own JSON object, with each router having "cost", "isLeafNode" and "children" as arguments.

`show ipv6 ospf6 graceful-restart helper [detail] [json]`
: This command shows the graceful-restart helper details including helper configuration parameters.

3.12.10 OSPFv3 Debugging

The following debug commands are supported:

`debug ospf6 abr`
: Toggle OSPFv3 ABR debugging messages.

`debug ospf6 asbr`
: Toggle OSPFv3 ASBR debugging messages.

`debug ospf6 border-routers {router-id [A.B.C.D] | area-id [A.B.C.D]}`
: Toggle OSPFv3 border router debugging messages. This can be specified for a router with specific Router-ID/Area-ID.

`debug ospf6 flooding`
: Toggle OSPFv3 flooding debugging messages.

`debug ospf6 interface`
: Toggle OSPFv3 interface related debugging messages.

`debug ospf6 lsa`
: Toggle OSPFv3 Link State Advertisements debugging messages.

debug ospf6 lsa aggregation
Toggle OSPFv3 Link State Advertisements summarization debugging messages.

debug ospf6 message
Toggle OSPFv3 message exchange debugging messages.

debug ospf6 neighbor
Toggle OSPFv3 neighbor interaction debugging messages.

debug ospf6 nssa
Toggle OSPFv3 Not So Stubby Area (NSSA) debugging messages.

debug ospf6 route
Toggle OSPFv3 routes debugging messages.

debug ospf6 spf
Toggle OSPFv3 Shortest Path calculation debugging messages.

debug ospf6 zebra
Toggle OSPFv3 zebra interaction debugging messages.

debug ospf6 graceful-restart
Toggle OSPFv3 graceful-restart helper debugging messages.

3.12.11 Sample configuration

Example of ospf6d configured on one interface and area:

```
interface eth0
 ipv6 ospf6 area 0.0.0.0
 ipv6 ospf6 instance-id 0
!
router ospf6
 ospf6 router-id 212.17.55.53
 area 0.0.0.0 range 2001:770:105:2::/64
!
```

Larger example with policy and various options set:

```
debug ospf6 neighbor state
!
interface fxp0
 ipv6 ospf6 area 0.0.0.0
 ipv6 ospf6 cost 1
 ipv6 ospf6 hello-interval 10
 ipv6 ospf6 dead-interval 40
 ipv6 ospf6 retransmit-interval 5
 ipv6 ospf6 priority 0
 ipv6 ospf6 transmit-delay 1
 ipv6 ospf6 instance-id 0
!
interface lo0
 ipv6 ospf6 cost 1
 ipv6 ospf6 hello-interval 10
 ipv6 ospf6 dead-interval 40
 ipv6 ospf6 retransmit-interval 5
```

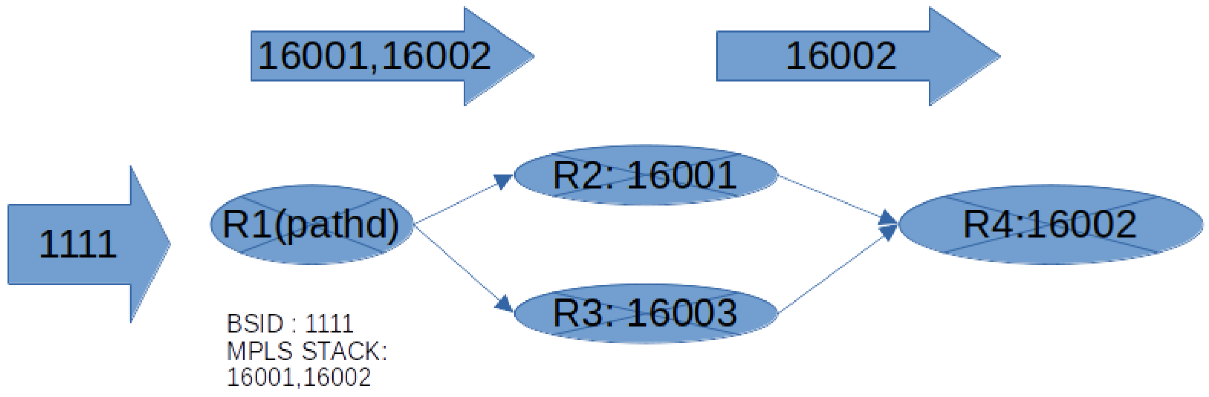
(continues on next page)

(continued from previous page)

```
 ipv6 ospf6 priority 1
 ipv6 ospf6 transmit-delay 1
 ipv6 ospf6 instance-id 0
!
router ospf6
 router-id 255.1.1.1
 redistribute static route-map static-ospf6
!
access-list access4 permit 127.0.0.1/32
!
ipv6 access-list access6 permit 3ffe:501::/32
ipv6 access-list access6 permit 2001:200::/48
ipv6 access-list access6 permit ::1/128
!
ipv6 prefix-list test-prefix seq 1000 deny any
!
route-map static-ospf6 permit 10
 match ipv6 address prefix-list test-prefix
 set metric-type type-2
 set metric 2000
!
line vty
 access-class access4
 ipv6 access-class access6
 exec-timeout 0 0
!
```

3.13 PATH

PATH is a daemon that handles the installation and deletion of Segment Routing (SR) Policies. Based on MPLS (This means that your OS of choice must support MPLS), SR add a stack of MPLS labels to ingress packets so these packets are egress through the desired path.



The SR policies and Segment Lists can be configured either locally by means of vtysh or centralized based on a SDN controller (ODL, Cisco, . . .) communicating using the PCEP protocol (**RFC 5440**).

3.13.1 Configuration

Explicit Segment Lists

This is the simplest way of configuration, no remote PCE is necessary. In order to create a config that match the graphics used in this documentation, we will create a segment list (SL) called SL1 with an element for each hop and that element will be assigned a MPLS label. Then the SL1 will be used in the policy `example1`, please note also the preference as in the case of multiple segment list it will be used with the criteria of bigger number more preference. Let see now the final configuration that match the graphics shown above.

```
segment-routing
 traffic-eng
  segment-list SL1
   index 10  mpls label 16001
   index 20  mpls label 16002
  !
  policy color 1 endpoint 192.0.2.4
   name example1
   binding-sid 1111
   candidate-path preference 100 name CP1 explicit segment-list SL1
```

Explicit Segment Lists and Traffic Engineering Database (TED)

Sometimes is difficult to know the values of MPLS labels (adjacency changes,…). Based on the support of IS-IS or OSPF we can activate TED support what will allow pathd to resolve MPLS based in different types of segments (:rfc: *draft-ietf-spring-segment-routing-policy-07*). The supported types are Type C (prefix and local interface), Type E (prefix and algorithm), Type F (a pair of IP's). So the configuration would change to this

```
segment-routing
 traffic-eng
  mpls-te on
  mpls-te import ospfv2
  segment-list SL1
   index 10  nai prefix 10.1.2.1/32 iface 1
   index 20  nai adjacency 10.1.20.1 10.1.20.2
  !
  policy color 1 endpoint 192.0.2.4
   name example1
   binding-sid 1111
   candidate-path preference 100 name CP1 explicit segment-list SL1
```

In this case no MPLS are provided but the pathd TED support will resolve the configuration provided to corresponding MPLS labels.

Note: Please note the `mpls-te` configuration added that activate the TED support and points to `ospfv2` so the ospfv2 (*Traffic Engineering*) daemon must be also running and configure to export TED information.

Note: It would be the same for isis (*Traffic Engineering*) but in the moment of writting it's not fully tested.

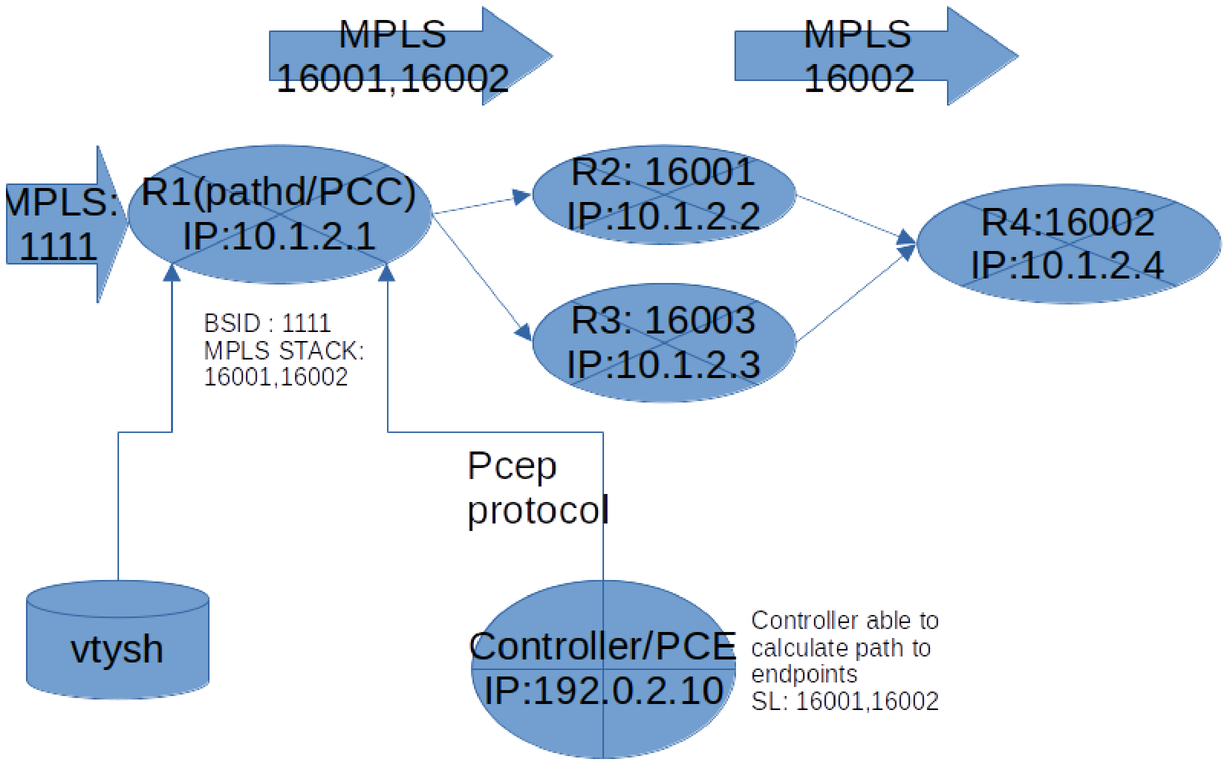
Dynamic Segment Lists

One of the useful options to configure is the creation of policies with the dynamic option. In this case based on a given endpoint the SL will be ,first calculated, and then sended by means of PCEP protocol by the configured PCE.

```
traffic-eng
 !
 pcep
  !
  pce PCE1
   address ip 192.0.2.10
  !
  pcc
   peer PCE1 precedence 10
 !
 policy color 1 endpoint 192.0.2.4
  name example
  binding-sid 1111
  candidate-path preference 100 name CP2 dynamic
```

Note: Please note the configuration for the remote pce which allows pathd to connect to the given PCE and act as a PCC (PCEP Client)

Note: If the TED support feature is active, the data obtained from PCE will be validated, so in a SL from PCEP/PCE the IP and MPLS will be checked against local TED obtained and built from the igp configured in that case.


MPLS 16001,16002
MPLS 16002
MPLS: 1111
R1(pathd/PCC) IP:10.1.2.1
R2: 16001 IP:10.1.2.2
R3: 16003 IP:10.1.2.3
R4:16002 IP:10.1.2.4
BSID : 1111
MPLS STACK: 16001,16002
Pcep protocol
vtysh
Controller/PCE IP:192.0.2.10
Controller able to calculate path to endpoints
SL: 16001,16002

Pce Initiated

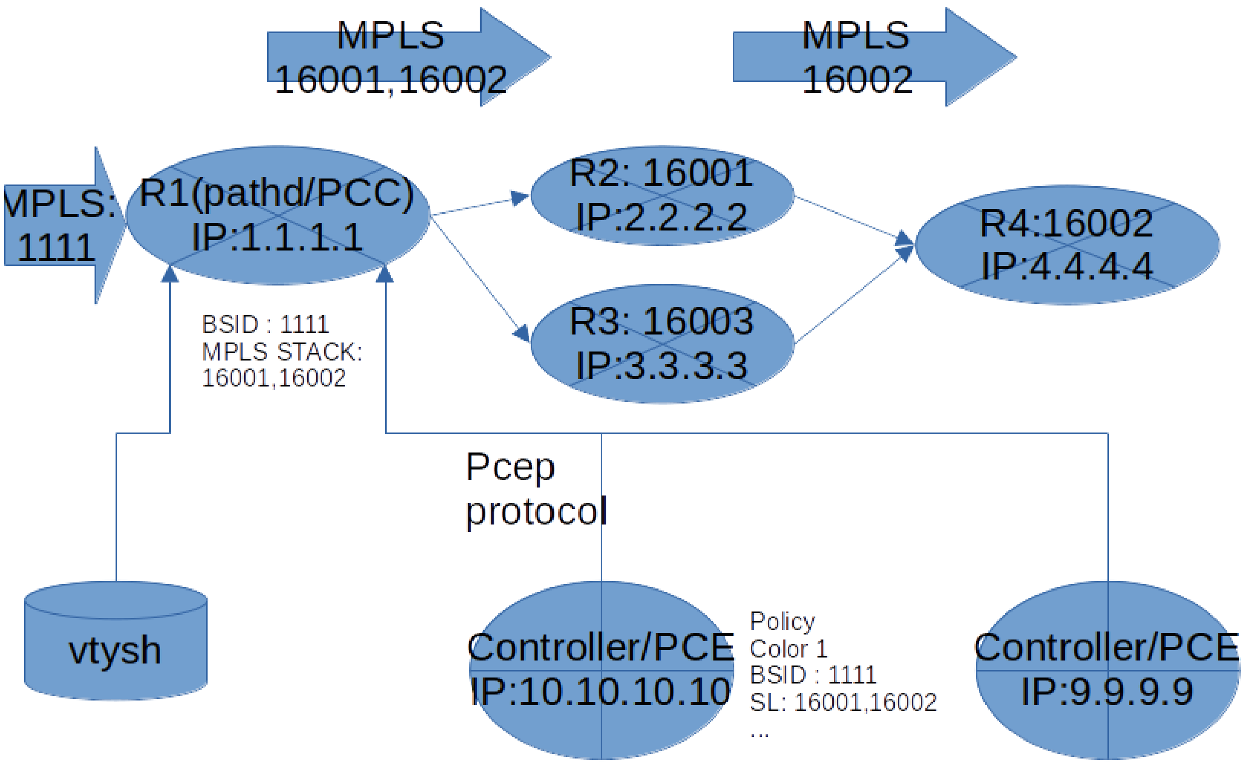
We can step forward in the use of our controller not only by asking to calculate paths to an endpoint but also to create the whole policies in the controller and obtain those by means of the PCEP protocol.

```
traffic-eng
 !
 pcep
  !
  pce PCE1
   address ip 192.0.2.10
   pce-initiated
  !
  pce PCE2
   address ip 192.0.2.9
   pce-initiated
  !
  pcc
   peer PCE1 precedence 10
   peer PCE2 precedence 20
 !
```

Note: Now there is no locally created policies in the config as they will be obtain from the configured pce. Please check command `show sr-te policy` in `vtysh` to see the obtained policies.

Note: Another interesting command is *`show mpls table`* to check the installed mpls configuration based in those obtained policies.

Note: SR Policies could be a mix of local, remote obtained from PCE and delegated to a PCE (but while testing Pce Initiated with Cisco PCE, happens that controller sends PCE initiated delete commands to delete the locally created configuration related to that PCE).



3.13.2 Starting

Default configuration file for *pathd* is `pathd.conf`. The typical location of `pathd.conf` is /etc/frr/pathd.conf.

If the user is using integrated config, then `pathd.conf` need not be present and the `frr.conf` is read instead.

PATH supports all the common FRR daemon start options which are documented elsewhere.

3.13.3 PCEP Support

A pceplib is included in the frr source tree and build by default.

To start pathd with pcep support the extra parameter *-M pathd_pcep* should be passed to the pathd daemon.

An example of command line with pcep module could be this

```
pathd -u root -g root -f pathd.conf -z /tmp/zebra-demo1.sock --vty_socket=/var/run/demo1.
↪vty -i /tmp/pathd-demo1.pid -M frr/modules/pathd_pcep.so --log file:/tmp/kk.txt
```

3.13.4 Pathd Configuration

Example:

```
debug pathd pcep basic
segment-routing
 traffic-eng
  mpls-te on
  mpls-te import ospfv2
  segment-list SL1
   index 10 mpls label 16010
   index 20 mpls label 16030
  !
  segment-list SL2
   index 10  nai prefix 10.1.2.1/32 iface 1
   index 20  nai adjacency 10.1.20.1 10.1.20.2
   index 30  nai prefix 10.10.10.5/32 algorithm 0
   index 40  mpls label 18001
  !
  policy color 1 endpoint 192.0.2.1
   name default
   binding-sid 4000
   candidate-path preference 100 name CP1 explicit segment-list SL1
   candidate-path preference 200 name CP2 dynamic
    affinity include-any 0x000000FF
    bandwidth 100000
    metric bound msd 16 required
    metric te 10
    objective-function mcp required
  !
  pcep
   pce-config GROUP1
    source-address 192.0.2.1
    tcp-md5-auth secret
    timer keep-alive 30
   !
   pce PCE1
    config GROUP1
    address ip 192.0.2.10
   !
   pce PCE2
    config GROUP1
    address ip 192.0.2.9
   !
   pcc
    peer PCE1 precedence 10
    peer PCE2 precedence 20
   !
  !
 !
!
```

Configuration Commands

`segment-routing`
Configure segment routing.

`traffic-eng`
Configure segment routing traffic engineering.

`mpls-te <on|off>`
Activate/Deactivate use of internal Traffic Engineering Database

`mpls-te import <ospfv2|ospfv3|isis>`
Load data from the selected igp

`segment-list NAME`
Delete or start a segment list definition.

`index INDEX mpls label LABEL`

`index INDEX nai adjacency A.B.C.D A.B.C.D`

`index INDEX nai prefix A.B.C.D/M algorithm <0|1>`

`index INDEX nai prefix A.B.C.D/M iface (0-65535)`
Delete or specify a segment in a segment list definition.

`policy color COLOR endpoint ENDPOINT`
Delete or start a policy definition.

`name NAME`
Specify the policy name.

`binding-sid LABEL`
Specify the policy SID.

`candidate-path preference PREFERENCE name NAME explicit segment-list SEGMENT-LIST-NAME`
Delete or define an explicit candidate path.

`candidate-path preference PREFERENCE name NAME dynamic`
Delete or start a dynamic candidate path definition.

`affinity <exclude-any|include-any|include-all> BITPATTERN`
Delete or specify an affinity constraint for a dynamic candidate path.

`bandwidth BANDWIDTH [required]`
Delete or specify a bandwidth constraint for a dynamic candidate path.

`metric [bound] METRIC VALUE [required]`
Delete or specify a metric constraint for a dynamic candidate path.

The possible metrics are:

- igp: IGP metric
- te: TE metric
- hc: Hop Counts
- abc: Aggregate bandwidth consumption
- mll: Load of the most loaded link
- igp: Cumulative IGP cost
- cte: Cumulative TE cost

- igp: P2MP IGP metric
- pte: P2MP TE metric
- phc: P2MP hop count metric
- msd: Segment-ID (SID) Depth
- pd: Path Delay metric
- pdv: Path Delay Variation metric
- pl: Path Loss metric
- ppd: P2MP Path Delay metric
- pdv: P2MP Path Delay variation metric
- ppl: P2MP Path Loss metric
- nap: Number of adaptations on a path
- nlp: Number of layers on a path
- dc: Domain Count metric
- bnc: Border Node Count metric

`objective-function OBJFUN1 [required]`

Delete or specify a PCEP objective function constraint for a dynamic candidate path.

The possible functions are:

- mcp: Minimum Cost Path [RFC5541]
- mlp: Minimum Load Path [RFC5541]
- mbp: Maximum residual Bandwidth Path [RFC5541]
- mbc: Minimize aggregate Bandwidth Consumption [RFC5541]
- mll: Minimize the Load of the most loaded Link [RFC5541]
- mcc: Minimize the Cumulative Cost of a set of paths [RFC5541]
- spt: Shortest Path Tree [RFC8306]
- mct: Minimum Cost Tree [RFC8306]
- mplp: Minimum Packet Loss Path [RFC8233]
- mup: Maximum Under-Utilized Path [RFC8233]
- mrup: Maximum Reserved Under-Utilized Path [RFC8233]
- mtd: Minimize the number of Transit Domains [RFC8685]
- mbn: Minimize the number of Border Nodes [RFC8685]
- mctd: Minimize the number of Common Transit Domains [RFC8685]
- msl: Minimize the number of Shared Links [RFC8800]
- mss: Minimize the number of Shared SRLGs [RFC8800]
- msn: Minimize the number of Shared Nodes [RFC8800]

`debug pathd pcep [basic|path|message|pceplib]`

Enable or disable debugging for the pcep module:

- basic: Enable basic PCEP logging

- path: Log the path structures
- message: Log the PCEP messages
- pceplib: Enable pceplib logging

`pcep`
: Configure PCEP support.

`pce-config NAME`
: Define a shared PCE configuration that can be used in multiple PCE declarations.

`pce NAME`
: Define or delete a PCE definition.

`config WORD`
: Select a shared configuration. If not defined, the default configuration will be used.

`address <ip A.B.C.D | ipv6 X:X::X:X> [port (1024-65535)]`
: Define the address and port of the PCE.

  If not specified, the port is the standard PCEP port 4189.

  This should be specified in the PCC peer definition.

`source-address [ip A.B.C.D | ipv6 X:X::X:X] [port PORT]`
: Define the address and/or port of the PCC as seen by the PCE. This can be used in a configuration group or a PCC peer declaration.

  If not specified, the source address will be the router identifier selected by zebra, and the port will be the standard PCEP port 4189.

  This can be specified in either the PCC peer definition or in a configuration group.

`tcp-md5-auth WORD`
: Enable TCP MD5 security with the given secret.

  This can be specified in either the PCC peer definition or in a configuration group.

`sr-draft07`
: Specify if a PCE only support segment routing draft 7, this flag will limit the PCC behavior to this draft.

  This can be specified in either the PCC peer definition or in a configuration group.

`pce-initiated`
: Specify if PCE-initiated LSP should be allowed for this PCE.

  This can be specified in either the PCC peer definition or in a configuration group.

`timer [keep-alive (1-63)] [min-peer-keep-alive (1-255)] [max-peer-keep-alive (1-255)] [dead-timer (4-25`
: Specify the PCEP timers.

  This can be specified in either the PCC peer definition or in a configuration group.

`pcc`
: Disable or start the definition of a PCC.

`msd (1-32)`
: Specify the maximum SID depth in a PCC definition.

`peer WORD [precedence (1-255)]`
: Specify a peer and its precedence in a PCC definition.

Debugging

debug pathd policy
Enable or disable Pathd policy information.

Introspection Commands

show sr-te policy [detail]
Display the segment routing policies.

```
router# show sr-te policy

 Endpoint  Color  Name     BSID  Status
 ------------------------------------------
 192.0.2.1   1      default  4000  Active
```

```
router# show sr-te policy detail

Endpoint: 192.0.2.1  Color: 1  Name: LOW_DELAY  BSID: 4000  Status: Active
    Preference: 100  Name: cand1  Type: explicit  Segment-List: sl1  Protocol-Origin:␣
↪Local
  * Preference: 200  Name: cand1  Type: dynamic  Segment-List: 32453452  Protocol-
↪Origin: PCEP
```

The asterisk (*) marks the best, e.g. active, candidate path. Note that for segment-lists which are retrieved via PCEP a random number based name is generated.

show sr-te pcep counters
Display the counters from pceplib.

show sr-te pcep pce-config [NAME]
Display a shared configuration. if no name is specified, the default configuration will be displayed.

show sr-te pcep pcc
Display PCC information.

show sr-te pcep session [NAME]
Display the information of a PCEP session, if not name is specified all the sessions will be displayed.

Utility Commands

clear sr-te pcep session [NAME]
Reset the pcep session by disconnecting from the PCE and performing the normal reconnection process. No configuration is changed.

3.13.5 Usage with BGP route-maps

It is possible to steer traffic 'into' a segment routing policy for routes learned through BGP using route-maps:

```
route-map SET_SR_POLICY permit 10
 set sr-te color 1
!
router bgp 1
 bgp router-id 192.0.2.2
 neighbor 192.0.2.1 remote-as 1
 neighbor 192.0.2.1 update-source lo
 !
 address-family ipv4 unicast
  neighbor 192.0.2.1 next-hop-self
  neighbor 192.0.2.1 route-map SET_SR_POLICY in
  redistribute static
 exit-address-family
 !
!
```

In this case, the SR Policy with color *1* and endpoint *192.0.2.1* is selected.

3.13.6 Sample configuration

```
! Default pathd configuration sample
!
password frr
log stdout

segment-routing
 traffic-eng
  segment-list test1
   index 10 mpls label 123
   index 20 mpls label 456
  !
  segment-list test2
   index 10 mpls label 321
   index 20 mpls label 654
  !
  policy color 1 endpoint 192.0.2.1
   name one
   binding-sid 100
   candidate-path preference 100 name test1 explicit segment-list test1
   candidate-path preference 200 name test2 explicit segment-list test2
  !
  policy color 2 endpoint 192.0.2.2
   name two
   binding-sid 101
   candidate-path preference 100 name def explicit segment-list test2
   candidate-path preference 200 name dyn dynamic
    bandwidth 12345
    metric bound abc 16 required
```

(continues on next page)

(continued from previous page)

```
    metric te 10
   !
  !
  pcep
   pcc-peer PCE1
    address ip 127.0.0.1
    sr-draft07
   !
   pcc
    peer PCE1
   !
 !
!
```

3.14 PIM

PIM – Protocol Independent Multicast

pimd supports pim-sm as well as igmp v2 and v3. pim is vrf aware and can work within the context of vrf's in order to do S,G mrouting. Additionally PIM can be used in the EVPN underlay network for optimizing forwarding of overlay BUM traffic.

Note: On Linux for PIM-SM operation you *must* have kernel version 4.19 or greater. To use PIM for EVPN BUM forwarding, kernels 5.0 or greater are required. OpenBSD has no multicast support and FreeBSD, and NetBSD only have support for SSM.

3.14.1 Starting and Stopping pimd

The default configuration file name of *pimd*'s is `pimd.conf`. When invoked *pimd* searches directory /etc/frr. If `pimd.conf` is not there then next search current directory.

pimd requires zebra for proper operation. Additionally *pimd* depends on routing properly setup and working in the network that it is working on.

```
# zebra -d
# pimd -d
```

Please note that *zebra* must be invoked before *pimd*.

To stop *pimd* please use:

```
kill `cat /var/run/frr/pimd.pid`
```

Certain signals have special meanings to *pimd*.

Signal	Meaning
SIGUSR1	Rotate the *pimd* logfile
SIGINT SIGTERM	*pimd* sweeps all installed PIM mroutes then terminates gracefully.

pimd invocation options. Common options that can be specified (*Common Invocation Options*).

`ip pim rp A.B.C.D A.B.C.D/M`

In order to use pim, it is necessary to configure a RP for join messages to be sent to. Currently the only methodology to do this is via static rp commands. All routers in the pim network must agree on these values. The first ip address is the RP's address and the second value is the matching prefix of group ranges covered. This command is vrf aware, to configure for a vrf, enter the vrf submode.

`ip pim rp keep-alive-timer (1-65535)`

Modify the time out value for a S,G flow from 1-65535 seconds at RP. The normal keepalive period for the KAT(S,G) defaults to 210 seconds. However, at the RP, the keepalive period must be at least the Register_Suppression_Time, or the RP may time out the (S,G) state before the next Null-Register arrives. Thus, the KAT(S,G) is set to max(Keepalive_Period, RP_Keepalive_Period) when a Register-Stop is sent. If choosing a value below 31 seconds be aware that some hardware platforms cannot see data flowing in better than 30 second chunks. This command is vrf aware, to configure for a vrf, enter the vrf submode.

`ip pim register-accept-list PLIST`

When pim receives a register packet the source of the packet will be compared to the prefix-list specified, PLIST, and if a permit is received normal processing continues. If a deny is returned for the source address of the register packet a register stop message is sent to the source.

`ip pim spt-switchover infinity-and-beyond [prefix-list PLIST]`

On the last hop router if it is desired to not switch over to the SPT tree configure this command. Optional parameter prefix-list can be use to control which groups to switch or not switch. If a group is PERMIT as per the PLIST, then the SPT switchover does not happen for it and if it is DENY, then the SPT switchover happens. This command is vrf aware, to configure for a vrf, enter the vrf submode.

`ip pim ecmp`

If pim has the a choice of ECMP nexthops for a particular RPF, pim will cause S,G flows to be spread out amongst the nexthops. If this command is not specified then the first nexthop found will be used. This command is vrf aware, to configure for a vrf, enter the vrf submode.

`ip pim ecmp rebalance`

If pim is using ECMP and an interface goes down, cause pim to rebalance all S,G flows across the remaining nexthops. If this command is not configured pim only modifies those S,G flows that were using the interface that went down. This command is vrf aware, to configure for a vrf, enter the vrf submode.

`ip pim join-prune-interval (1-65535)`

Modify the join/prune interval that pim uses to the new value. Time is specified in seconds. This command is vrf aware, to configure for a vrf, enter the vrf submode. The default time is 60 seconds. If you enter a value smaller than 60 seconds be aware that this can and will affect convergence at scale.

`ip pim keep-alive-timer (1-65535)`

Modify the time out value for a S,G flow from 1-65535 seconds. If choosing a value below 31 seconds be aware that some hardware platforms cannot see data flowing in better than 30 second chunks. This command is vrf aware, to configure for a vrf, enter the vrf submode.

`ip pim packets (1-255)`

When processing packets from a neighbor process the number of packets incoming at one time before moving on to the next task. The default value is 3 packets. This command is only useful at scale when you can possibly have a large number of pim control packets flowing. This command is vrf aware, to configure for a vrf, enter the vrf submode.

`ip pim register-suppress-time (1-65535)`

Modify the time that pim will register suppress a FHR will send register notifications to the kernel. This command is vrf aware, to configure for a vrf, enter the vrf submode.

`ip pim send-v6-secondary`

When sending pim hello packets tell pim to send any v6 secondary addresses on the interface. This information is used to allow pim to use v6 nexthops in it's decision for RPF lookup. This command is vrf aware, to configure for a vrf, enter the vrf submode.

`ip pim ssm prefix-list WORD`
Specify a range of group addresses via a prefix-list that forces pim to never do SM over. This command is vrf aware, to configure for a vrf, enter the vrf submode.

`ip multicast rpf-lookup-mode WORD`
Modify how PIM does RPF lookups in the zebra routing table. You can use these choices:

longer-prefix Lookup the RPF in both tables using the longer prefix as a match

lower-distance Lookup the RPF in both tables using the lower distance as a match

mrib-only Lookup in the Multicast RIB only

mrib-then-urib Lookup in the Multicast RIB then the Unicast Rib, returning first found. This is the default value for lookup if this command is not entered

urib-only Lookup in the Unicast Rib only.

`ip igmp generate-query-once [version (2-3)]`
Generate IGMP query (v2/v3) on user requirement. This will not depend on the existing IGMP general query timer.If no version is provided in the cli, the default will be the igmp version enabled on that interface.

`ip igmp watermark-warn (1-65535)`
Configure watermark warning generation for an igmp group limit. Generates warning once the configured group limit is reached while adding new groups. 'no' form of the command disables the warning generation. This command is vrf aware. To configure per vrf, enter vrf submode.

3.14.2 PIM Interface Configuration

PIM interface commands allow you to configure an interface as either a Receiver or a interface that you would like to form pim neighbors on. If the interface is in a vrf, enter the interface command with the vrf keyword at the end.

`ip pim active-active`
Turn on pim active-active configuration for a Vxlan interface. This command will not do anything if you do not have the underlying ability of a mlag implementation.

`ip pim bsm`
Tell pim that we would like to use this interface to process bootstrap messages. This is enabled by default. 'no' form of this command is used to restrict bsm messages on this interface.

`ip pim unicast-bsm`
Tell pim that we would like to allow interface to process unicast bootstrap messages. This is enabled by default. 'no' form of this command is used to restrict processing of unicast bsm messages on this interface.

`ip pim drpriority (1-4294967295)`
Set the DR Priority for the interface. This command is useful to allow the user to influence what node becomes the DR for a lan segment.

`ip pim hello (1-65535) (1-65535)`
Set the pim hello and hold interval for a interface.

`ip pim`
Tell pim that we would like to use this interface to form pim neighbors over. Please note that this command does not enable the reception of IGMP reports on the interface. Refer to the next *ip igmp* command for IGMP management.

`ip pim use-source A.B.C.D`
If you have multiple addresses configured on a particular interface and would like pim to use a specific source address associated with that interface.

`ip pim passive`
    Disable sending and receiving pim control packets on the interface.

`ip igmp`
    Tell pim to receive IGMP reports and Query on this interface. The default version is v3. This command is useful on a LHR.

`ip igmp join A.B.C.D [A.B.C.D]`
    Join multicast group or source-group on an interface.

`ip igmp query-interval (1-65535)`
    Set the IGMP query interval that PIM will use.

`ip igmp query-max-response-time (1-65535)`
    Set the IGMP query response timeout value. If an report is not returned in the specified time we will assume the S,G or *,G has timed out.

`ip igmp version (2-3)`
    Set the IGMP version used on this interface. The default value is 3.

`ip multicast boundary oil WORD`
    Set a pim multicast boundary, based upon the WORD prefix-list. If a pim join or IGMP report is received on this interface and the Group is denied by the prefix-list, PIM will ignore the join or report.

`ip igmp last-member-query-count (1-255)`
    Set the IGMP last member query count. The default value is 2. 'no' form of this command is used to to configure back to the default value.

`ip igmp last-member-query-interval (1-65535)`
    Set the IGMP last member query interval in deciseconds. The default value is 10 deciseconds. 'no' form of this command is used to to configure back to the default value.

`ip mroute INTERFACE A.B.C.D [A.B.C.D]`
    Set a static multicast route for a traffic coming on the current interface to be forwarded on the given interface if the traffic matches the group address and optionally the source address.

See also:

PIM BFD Configuration

3.14.3 PIM Multicast RIB

In order to influence Multicast RPF lookup, it is possible to insert into zebra routes for the Multicast RIB. These routes are only used for RPF lookup and will not be used by zebra for insertion into the kernel *or* for normal rib processing. As such it is possible to create weird states with these commands. Use with caution. Most of the time this will not be necessary.

`ip mroute A.B.C.D/M A.B.C.D (1-255)`
    Insert into the Multicast Rib Route A.B.C.D/M with specified nexthop. The distance can be specified as well if desired.

`ip mroute A.B.C.D/M INTERFACE (1-255)`
    Insert into the Multicast Rib Route A.B.C.D/M using the specified INTERFACE. The distance can be specified as well if desired.

3.14.4 Multicast Source Discovery Protocol (MSDP) Configuration

MSDP can be setup in different ways:

- MSDP meshed-group: where all peers are connected with each other creating a fully meshed network. SAs (source active) messages are not forwarded in this mode because the origin is able to send SAs to all members.

  This setup is commonly used with anycast.

- MSDP peering: when there is one or more peers that are not fully meshed. SAs may be forwarded depending on the result of filtering and RPF checks.

  This setup is commonly consistent with BGP peerings (for RPF checks).

- MSDP default peer: there is only one peer and all SAs will be forwarded there.

Note: MSDP default peer and SA filtering is not implemented.

Commands available for MSDP:

`ip msdp timers (1-65535) (1-65535) [(1-65535)]`
: Configure global MSDP timers.

  First value is the keep-alive interval. This configures the interval in seconds between keep-alive messages. The default value is 60 seconds. It should be less than the remote hold time.

  Second value is the hold-time. This configures the interval in seconds before closing a non responding connection. The default value is 75. This value should be greater than the remote keep alive time.

  Third value is the connection retry interval and it is optional. This configures the interval between connection attempts. The default value is 30 seconds.

`ip msdp mesh-group WORD member A.B.C.D`
: Create or update a mesh group to include the specified MSDP peer.

`ip msdp mesh-group WORD source A.B.C.D`
: Create or update a mesh group to set the source address used to connect to peers.

`ip msdp peer A.B.C.D source A.B.C.D`
: Create a regular MSDP session with peer using the specified source address.

3.14.5 Show PIM Information

All PIM show commands are vrf aware and typically allow you to insert a specified vrf command if information is desired about a specific vrf. If no vrf is specified then the default vrf is assumed. Finally the special keyword 'all' allows you to look at all vrfs for the command. Naming a vrf 'all' will cause great confusion.

`show ip igmp interface`
: Display IGMP interface information.

`show ip igmp [vrf NAME] join [json]`
: Display IGMP static join information for a specific vrf.

`show ip igmp [vrf NAME$vrf_name] groups [INTERFACE$ifname [GROUP$grp_str]] [detail] [json$json]`
: Display IGMP static join information for all the vrfs present.

`show ip igmp vrf all groups [GROUP$grp_str] [detail$detail] [json$json]`
: Display IGMP groups information.

`show ip igmp groups retransmissions`
: Display IGMP group retransmission information.

`show ip igmp [vrf NAME] sources [json]`
    Display IGMP sources information.

`show ip igmp sources retransmissions`
    Display IGMP source retransmission information.

`show ip igmp statistics`
    Display IGMP statistics information.

`show ip multicast`
    Display various information about the interfaces used in this pim instance.

`show ip mroute [vrf NAME] [A.B.C.D [A.B.C.D]] [fill] [json]`
    Display information about installed into the kernel S,G mroutes. If one address is specified we assume it is the Group we are interested in displaying data on. If the second address is specified then it is Source Group. The keyword *fill* says to fill in all assumed data for test/data gathering purposes.

`show ip mroute [vrf NAME] count [json]`
    Display information about installed into the kernel S,G mroutes and in addition display data about packet flow for the mroutes for a specific vrf.

`show ip mroute vrf all count [json]`
    Display information about installed into the kernel S,G mroutes and in addition display data about packet flow for the mroutes for all vrfs.

`show ip mroute [vrf NAME] summary [json]`
    Display total number of S,G mroutes and number of S,G mroutes installed into the kernel for a specific vrf.

`show ip mroute vrf all summary [json]`
    Display total number of S,G mroutes and number of S,G mroutes installed into the kernel for all vrfs.

`show ip msdp mesh-group`
    Display the configured mesh-groups, the local address associated with each mesh-group, the peer members included in each mesh-group, and their status.

`show ip msdp peer`
    Display information about the MSDP peers. That includes the peer address, the local address used to establish the connection to the peer, the connection status, and the number of active sources.

`show ip pim assert`
    Display information about asserts in the PIM system for S,G mroutes. This command does not show S,G Channel states that in a NOINFO state.

`show ip pim assert-internal`
    Display internal assert state for S,G mroutes

`show ip pim assert-metric`
    Display metric information about assert state for S,G mroutes

`show ip pim assert-winner-metric`
    Display winner metric for assert state for S,G mroutes

`show ip pim group-type`
    Display SSM group ranges.

`show ip pim interface`
    Display information about interfaces PIM is using.

`show ip pim mlag [vrf NAME|all] interface [detail|WORD] [json]`
    Display mlag interface information.

show ip pim join
Display information about PIM joins received. If one address is specified then we assume it is the Group we are interested in displaying data on. If the second address is specified then it is Source Group.

show ip pim local-membership
Display information about PIM interface local-membership.

show ip pim mlag summary [json]
Display mlag information state that PIM is keeping track of.

show ip pim neighbor
Display information about PIM neighbors.

show ip pim nexthop
Display information about pim nexthops that are being used.

show ip pim nexthop-lookup
Display information about a S,G pair and how the RPF would be chosen. This is especially useful if there are ECMP's available from the RPF lookup.

show ip pim [vrf NAME] rp-info [A.B.C.D/M] [json]
Display information about RP's that are configured on this router.

You can filter the output by specifying an arbitrary group range.

```
# show ip pim rp-info
RP address       group/prefix-list   OIF               I am RP    Source   Group-
↪Type
192.168.10.123   225.0.0.0/24        eth2              yes        Static   ASM
192.168.10.123   239.0.0.0/8         eth2              yes        Static   ASM
192.168.10.123   239.4.0.0/24        eth2              yes        Static   SSM

# show ip pim rp-info 239.4.0.0/25
RP address       group/prefix-list   OIF               I am RP    Source   Group-
↪Type
192.168.10.123   239.0.0.0/8         eth2              yes        Static   ASM
192.168.10.123   239.4.0.0/24        eth2              yes        Static   SSM
```

show ip pim rpf
Display information about currently being used S,G's and their RPF lookup information. Additionally display some statistics about what has been happening on the router.

show ip pim secondary
Display information about an interface and all the secondary addresses associated with it.

show ip pim state
Display information about known S,G's and incoming interface as well as the OIL and how they were chosen.

show ip pim [vrf NAME] upstream [A.B.C.D [A.B.C.D]] [json]
Display upstream information about a S,G mroute. Allow the user to specify sub Source and Groups that we are only interested in.

show ip pim upstream-join-desired
Display upstream information for S,G's and if we desire to join the multicast tree

show ip pim upstream-rpf
Display upstream information for S,G's and the RPF data associated with them.

show ip pim [vrf NAME] mlag upstream [A.B.C.D [A.B.C.D]] [json]
Display upstream entries that are synced across MLAG switches. Allow the user to specify sub Source and Groups address filters.

`show ip pim mlag summary`
: Display PIM MLAG (multi-chassis link aggregation) session status and control message statistics.

`show ip pim bsr`
: Display current bsr, its uptime and last received bsm age.

`show ip pim bsrp-info`
: Display group-to-rp mappings received from E-BSR.

`show ip pim bsm-database`
: Display all fragments of stored bootstrap message in user readable format.

`mtrace A.B.C.D [A.B.C.D]`
: Display multicast traceroute towards source, optionally for particular group.

`show ip multicast count [vrf NAME] [json]`
: Display multicast data packets count per interface for a vrf.

`show ip multicast count vrf all [json]`
: Display multicast data packets count per interface for all vrf.

See also:

Multicast RIB Commands

3.14.6 PIM Debug Commands

The debugging subsystem for PIM behaves in accordance with how FRR handles debugging. You can specify debugging at the enable CLI mode as well as the configure CLI mode. If you specify debug commands in the configuration cli mode, the debug commands can be persistent across restarts of the FRR pimd if the config was written out.

`debug igmp`
: This turns on debugging for IGMP protocol activity.

`debug mtrace`
: This turns on debugging for mtrace protocol activity.

`debug mroute`
: This turns on debugging for PIM interaction with kernel MFC cache.

`debug pim events`
: This turns on debugging for PIM system events. Especially timers.

`debug pim nht`
: This turns on debugging for PIM nexthop tracking. It will display information about RPF lookups and information about when a nexthop changes.

`debug pim nht detail`
: This turns on debugging for PIM nexthop in detail. This is not enabled by default.

`debug pim packet-dump`
: This turns on an extraordinary amount of data. Each pim packet sent and received is dumped for debugging purposes. This should be considered a developer only command.

`debug pim packets`
: This turns on information about packet generation for sending and about packet handling from a received packet.

`debug pim trace`
: This traces pim code and how it is running.

`debug pim bsm`
: This turns on debugging for BSR message processing.

debug pim zebra
This gathers data about events from zebra that come up through the ZAPI.

3.14.7 PIM Clear Commands

Clear commands reset various variables.

clear ip interfaces
Reset interfaces.

clear ip igmp interfaces
Reset IGMP interfaces.

clear ip mroute
Reset multicast routes.

clear ip mroute [vrf NAME] count
When this command is issued, reset the counts of data shown for packet count, byte count and wrong interface to 0 and start count up from this spot.

clear ip pim interfaces
Reset PIM interfaces.

clear ip pim oil
Rescan PIM OIL (output interface list).

clear ip pim [vrf NAME] bsr-data
This command will clear the BSM scope data struct. This command also removes the next hop tracking for the bsr and resets the upstreams for the dynamically learnt RPs.

3.14.8 PIM EVPN configuration

To use PIM in the underlay for overlay BUM forwarding associate a multicast group with the L2 VNI. The actual configuration is based on your distribution. Here is an ifupdown2 example:

```
auto vx-10100
iface vx-10100
    vxlan-id 10100
    bridge-access 100
    vxlan-local-tunnelip 27.0.0.11
    vxlan-mcastgrp 239.1.1.100
```

Note: PIM will see the `vxlan-mcastgrp` configuration and auto configure state to properly forward BUM traffic.

PIM also needs to be configured in the underlay to allow the BUM MDT to be setup. This is existing PIM configuration:

- Enable pim on the underlay L3 interface via the “ip pim” command.
- Configure RPs for the BUM multicast group range.
- Ensure the PIM is enabled on the lo of the VTEPs and the RP.

3.14.9 Sample configuration

```
debug igmp
debug pim
debug pim zebra

! You may want to enable ssmpingd for troubleshooting
! See http://www.venaas.no/multicast/ssmping/
!
ip ssmpingd 1.1.1.1
ip ssmpingd 2.2.2.2

! HINTS:
!  - Enable "ip pim ssm" on the interface directly attached to the
!    multicast source host (if this is the first-hop router)
!  - Enable "ip pim ssm" on pim-routers-facing interfaces
!  - Enable "ip igmp" on IGMPv3-hosts-facing interfaces
!  - In order to inject IGMPv3 local membership information in the
!    PIM protocol state, enable both "ip pim ssm" and "ip igmp" on
!    the same interface; otherwise PIM won't advertise
!    IGMPv3-learned membership to other PIM routers

interface eth0
 ip pim ssm
 ip igmp
```

3.15 PIMv6

PIMv6 – Protocol Independent Multicast for IPv6

pim6d supports pim-sm as well as MLD v1 and v2. PIMv6 is vrf aware and can work within the context of vrf's in order to do S,G mrouting.

3.15.1 Starting and Stopping pim6d

The default configuration file name of *pim6d*'s is `pim6d.conf`. When invoked *pim6d* searches directory /etc/frr. If `pim6d.conf` is not there then next search current directory.

pim6d requires zebra for proper operation. Additionally *pim6d* depends on routing properly setup and working in the network that it is working on.

```
# zebra -d
# pim6d -d
```

Please note that *zebra* must be invoked before *pim6d*.

To stop *pim6d* please use:

```
kill `cat /var/run/frr/pim6d.pid`
```

Certain signals have special meanings to *pim6d*.

Signal	Meaning
SIGUSR1	Rotate the *pim6d* logfile
SIGINT SIGTERM	*pim6d* sweeps all installed PIM mroutes then terminates gracefully.

pim6d invocation options. Common options that can be specified (*Common Invocation Options*).

`ipv6 pim rp X:X::X:X Y:Y::Y:Y/M`
In order to use pimv6, it is necessary to configure a RP for join messages to be sent to. Currently the only methodology to do this is via static rp commands. All routers in the pimv6 network must agree on these values. The first ipv6 address is the RP's address and the second value is the matching prefix of group ranges covered. This command is vrf aware, to configure for a vrf, enter the vrf submode.

`ipv6 pim rp X:X::X:X prefix-list WORD`
This CLI helps in configuring RP address for a range of groups specified by the prefix-list.

`ipv6 pim rp keep-alive-timer (1-65535)`
Modify the time out value for a S,G flow from 1-65535 seconds at RP. The normal keepalive period for the KAT(S,G) defaults to 210 seconds. However, at the RP, the keepalive period must be at least the Register_Suppression_Time, or the RP may time out the (S,G) state before the next Null-Register arrives. Thus, the KAT(S,G) is set to max(Keepalive_Period, RP_Keepalive_Period) when a Register-Stop is sent. If choosing a value below 31 seconds be aware that some hardware platforms cannot see data flowing in better than 30 second chunks. This command is vrf aware, to configure for a vrf, enter the vrf submode.

`ipv6 pim spt-switchover infinity-and-beyond [prefix-list PLIST]`
On the last hop router if it is desired to not switch over to the SPT tree configure this command. Optional parameter prefix-list can be use to control which groups to switch or not switch. If a group is PERMIT as per the PLIST, then the SPT switchover does not happen for it and if it is DENY, then the SPT switchover happens. This command is vrf aware, to configure for a vrf, enter the vrf submode.

`ipv6 pim join-prune-interval (1-65535)`
Modify the join/prune interval that pim uses to the new value. Time is specified in seconds. This command is vrf aware, to configure for a vrf, enter the vrf submode. The default time is 60 seconds. If you enter a value smaller than 60 seconds be aware that this can and will affect convergence at scale.

`ipv6 pim keep-alive-timer (1-65535)`
Modify the time out value for a S,G flow from 1-65535 seconds. If choosing a value below 31 seconds be aware that some hardware platforms cannot see data flowing in better than 30 second chunks. This command is vrf aware, to configure for a vrf, enter the vrf submode.

`ipv6 pim packets (1-255)`
When processing packets from a neighbor process the number of packets incoming at one time before moving on to the next task. The default value is 3 packets. This command is only useful at scale when you can possibly have a large number of pim control packets flowing. This command is vrf aware, to configure for a vrf, enter the vrf submode.

`ipv6 pim register-suppress-time (1-65535)`
Modify the time that pim will register suppress a FHR will send register notifications to the kernel. This command is vrf aware, to configure for a vrf, enter the vrf submode.

`ipv6 ssmpingd [X:X::X:X]`
Enable ipv6 ssmpingd configuration. A network level management tool to check whether one can receive multicast packets via SSM from host. The host target given to ssmping must run the ssmpingd daemon which listens for IPv4 and IPv6 unicast requests. When it receives one, it responds to a well known SSM multicast group which ssmping just have joined.

3.15.2 PIMv6 Interface Configuration

PIMv6 interface commands allow you to configure an interface as either a Receiver or a interface that you would like to form pimv6 neighbors on. If the interface is in a vrf, enter the interface command with the vrf keyword at the end.

`ipv6 pim active-active`
: Turn on pim active-active configuration for a Vxlan interface. This command will not do anything if you do not have the underlying ability of a mlag implementation.

`ipv6 pim drpriority (1-4294967295)`
: Set the DR Priority for the interface. This command is useful to allow the user to influence what node becomes the DR for a lan segment.

`ipv6 pim hello (1-65535) (1-65535)`
: Set the pim hello and hold interval for a interface.

`ipv6 pim`
: Tell pim that we would like to use this interface to form pim neighbors over. Please note that this command does not enable the reception of MLD reports on the interface. Refer to the next `ipv6 mld` command for MLD management.

`ipv6 pim use-source X:X::X:X`
: If you have multiple addresses configured on a particular interface and would like pim to use a specific source address associated with that interface.

`ipv6 pim passive`
: Disable sending and receiving pim control packets on the interface.

`ipv6 pim bsm`
: Tell pim that we would like to use this interface to process bootstrap messages. This is enabled by default. 'no' form of this command is used to restrict bsm messages on this interface.

`ipv6 pim unicast-bsm`
: Tell pim that we would like to allow interface to process unicast bootstrap messages. This is enabled by default. 'no' form of this command is used to restrict processing of unicast bsm messages on this interface.

`ipv6 mld`
: Tell pim to receive MLD reports and Query on this interface. The default version is v2. This command is useful on a LHR.

`ipv6 mld join X:X::X:X [Y:Y::Y:Y]`
: Join multicast group or source-group on an interface.

`ipv6 mld query-interval (1-65535)`
: Set the MLD query interval that PIM will use.

`ipv6 mld query-max-response-time (1-65535)`
: Set the MLD query response timeout value. If an report is not returned in the specified time we will assume the S,G or *,G has timed out.

`ipv6 mld version (1-2)`
: Set the MLD version used on this interface. The default value is 2.

`ipv6 multicast boundary oil WORD`
: Set a PIMv6 multicast boundary, based upon the WORD prefix-list. If a PIMv6 join or MLD report is received on this interface and the Group is denied by the prefix-list, PIMv6 will ignore the join or report.

`ipv6 mld last-member-query-count (1-255)`
: Set the MLD last member query count. The default value is 2. 'no' form of this command is used to configure back to the default value.

ipv6 mld last-member-query-interval (1-65535)
Set the MLD last member query interval in deciseconds. The default value is 10 deciseconds. 'no' form of this command is used to to configure back to the default value.

ipv6 mroute INTERFACE X:X::X:X [Y:Y::Y:Y]
Set a static multicast route for a traffic coming on the current interface to be forwarded on the given interface if the traffic matches the group address and optionally the source address.

3.15.3 Show PIMv6 Information

All PIMv6 show commands are vrf aware and typically allow you to insert a specified vrf command if information is desired about a specific vrf. If no vrf is specified then the default vrf is assumed. Finally the special keyword 'all' allows you to look at all vrfs for the command. Naming a vrf 'all' will cause great confusion.

PIM protocol state

show ipv6 pim [vrf NAME] group-type [json]
Display SSM group ranges.

show ipv6 pim interface
Display information about interfaces PIM is using.

show ipv6 pim [vrf NAME] join [X:X::X:X [X:X::X:X]] [json]

show ipv6 pim vrf all join [json]
Display information about PIM joins received. If one address is specified then we assume it is the Group we are interested in displaying data on. If the second address is specified then it is Source Group.

show ipv6 pim [vrf NAME] local-membership [json]
Display information about PIM interface local-membership.

show ipv6 pim [vrf NAME] neighbor [detail|WORD] [json]

show ipv6 pim vrf all neighbor [detail|WORD] [json]
Display information about PIM neighbors.

show ipv6 pim [vrf NAME] nexthop
Display information about pim nexthops that are being used.

show ipv6 pim [vrf NAME] nexthop-lookup X:X::X:X X:X::X:X
Display information about a S,G pair and how the RPF would be chosen. This is especially useful if there are ECMP's available from the RPF lookup.

show ipv6 pim [vrf NAME] rp-info [json]

show ipv6 pim vrf all rp-info [json]
Display information about RP's that are configured on this router.

show ipv6 pim [vrf NAME] rpf [json]

show ipv6 pim vrf all rpf [json]
Display information about currently being used S,G's and their RPF lookup information. Additionally display some statistics about what has been happening on the router.

show ipv6 pim [vrf NAME] secondary
Display information about an interface and all the secondary addresses associated with it.

show ipv6 pim [vrf NAME] state [X:X::X:X [X:X::X:X]] [json]

show ipv6 pim vrf all state [X:X::X:X [X:X::X:X]] [json]
Display information about known S,G's and incoming interface as well as the OIL and how they were chosen.

show ipv6 pim [vrf NAME] upstream [X:X::X:X [Y:Y::Y:Y]] [json]

show ipv6 pim vrf all upstream [json]
Display upstream information about a S,G mroute. Allow the user to specify sub Source and Groups that we are interested in.

show ipv6 pim [vrf NAME] upstream-join-desired [json]
Display upstream information for S,G's and if we desire to join the multicast tree

show ipv6 pim [vrf NAME] upstream-rpf [json]
Display upstream information for S,G's and the RPF data associated with them.

show ipv6 pim [vrf NAME] interface traffic [WORD] [json]
Display information about the number of PIM protocol packets sent/received on an interface.

MLD state

show ipv6 mld [vrf NAME] interface [IFNAME] [detail|json]
Display per-interface MLD state, elected querier and related timers. Use the `detail` or `json` options for further information (the JSON output always contains all details.)

show ipv6 mld [vrf NAME] statistics [interface IFNAME] [json]
Display packet and error counters for MLD interfaces. All counters are packet counters (not bytes) and wrap at 64 bit. In some rare cases, malformed received MLD reports may be partially processed and counted on multiple counters.

show ipv6 mld [vrf NAME] joins [{interface IFNAME|groups X:X::X:X/M|sources X:X::X:X/M|detail}] [json]
Display joined groups tracked by MLD. `interface`, `groups` and `sources` options may be used to limit output to a subset (note `sources` refers to the multicast traffic sender, not the host that joined to receive the traffic.)

The `detail` option also reports which hosts have joined (subscribed) to particular `S,G`. This information is only available for MLDv2 hosts with a MLDv2 querier. MLDv1 joins are recorded as "untracked" and shown in the `NonTrkSeen` output column.

show ipv6 mld [vrf NAME] groups [json]
Display MLD group information.

General multicast routing state

show ipv6 multicast
Display various information about the interfaces used in this pim instance.

show ipv6 multicast count [vrf NAME] [json]
Display multicast data packets count per interface for a vrf.

show ipv6 multicast count vrf all [json]
Display multicast data packets count per interface for all vrf.

show ipv6 mroute [vrf NAME] [X:X::X:X [X:X::X:X]] [fill] [json]
Display information about installed into the kernel S,G mroutes. If one address is specified we assume it is the Group we are interested in displaying data on. If the second address is specified then it is Source Group. The keyword `fill` says to fill in all assumed data for test/data gathering purposes.

`show ipv6 mroute [vrf NAME] count [json]`
Display information about installed into the kernel S,G mroutes and in addition display data about packet flow for the mroutes for a specific vrf.

`show ipv6 mroute vrf all count [json]`
Display information about installed into the kernel S,G mroutes and in addition display data about packet flow for the mroutes for all vrfs.

`show ipv6 mroute [vrf NAME] summary [json]`
Display total number of S,G mroutes and number of S,G mroutes installed into the kernel for a specific vrf.

`show ipv6 mroute vrf all summary [json]`
Display total number of S,G mroutes and number of S,G mroutes installed into the kernel for all vrfs.

`show ipv6 pim bsr`
Display current bsr, its uptime and last received bsm age.

`show ipv6 pim bsrp-info`
Display group-to-rp mappings received from E-BSR.

`show ipv6 pim bsm-database`
Display all fragments of stored bootstrap message in user readable format.

3.15.4 PIMv6 Clear Commands

Clear commands reset various variables.

`clear ipv6 mroute`
Reset multicast routes.

`clear ipv6 mroute [vrf NAME] count`
When this command is issued, reset the counts of data shown for packet count, byte count and wrong interface to 0 and start count up from this spot.

`clear ipv6 pim interfaces`
Reset PIMv6 interfaces.

`clear ipv6 pim [vrf NAME] interface traffic`
When this command is issued, resets the information about the number of PIM protocol packets sent/received on an interface.

`clear ipv6 pim oil`
Rescan PIMv6 OIL (output interface list).

`clear ipv6 pim [vrf NAME] bsr-data`
This command will clear the BSM scope data struct. This command also removes the next hop tracking for the bsr and resets the upstreams for the dynamically learnt RPs.

3.15.5 PIMv6 Debug Commands

The debugging subsystem for PIMv6 behaves in accordance with how FRR handles debugging. You can specify debugging at the enable CLI mode as well as the configure CLI mode. If you specify debug commands in the configuration cli mode, the debug commands can be persistent across restarts of the FRR pim6d if the config was written out.

`debug mld`
This turns on debugging for MLD protocol activity.

`debug pimv6 events`
This turns on debugging for PIMv6 system events. Especially timers.

debug pimv6 nht
This turns on debugging for PIMv6 nexthop tracking. It will display information about RPF lookups and information about when a nexthop changes.

debug pimv6 nht detail
This turns on debugging for PIMv6 nexthop in detail. This is not enabled by default.

debug pimv6 packet-dump
This turns on an extraordinary amount of data. Each pim packet sent and received is dumped for debugging purposes. This should be considered a developer only command.

debug pimv6 packets
This turns on information about packet generation for sending and about packet handling from a received packet.

debug pimv6 trace
This traces pim code and how it is running.

debug pimv6 zebra
This gathers data about events from zebra that come up through the ZAPI.

debug mroute6
This turns on debugging for PIMv6 interaction with kernel MFC cache.

debug mroute6 detail
This turns on detailed debugging for PIMv6 interaction with kernel MFC cache.

debug mld events
This turns on debugging for MLD system events.

debug mld packets
This turns on information about MLD protocol packets handling.

debug mld trace [detail]
This traces mld code and how it is running.

debug pimv6 bsm
This turns on debugging for BSR message processing.

3.16 PBR

PBR is Policy Based Routing. This implementation supports a very simple interface to allow admins to influence routing on their router. At this time you can only match on destination and source prefixes for an incoming interface. At this point in time, this implementation will only work on Linux.

3.16.1 Starting PBR

Default configuration file for *pbrd* is `pbrd.conf`. The typical location of `pbrd.conf` is /etc/frr/pbrd.conf.

If the user is using integrated config, then `pbrd.conf` need not be present and the `frr.conf` is read instead.

PBR supports all the common FRR daemon start options which are documented elsewhere.

3.16.2 Nexthop Groups

Nexthop groups are a way to encapsulate ECMP information together. It's a listing of ECMP nexthops used to forward packets for when a pbr-map is matched. For detailed instructions on how to specify a nexthop group on the CLI, see the nexthop-groups section.

Showing Nexthop Group Information

`show pbr nexthop-groups [NAME] [json]`

Display information on a PBR nexthop-group. If `NAME` is omitted, all nexthop groups are shown. Setting `json` will provide the same information in an array of objects which obey the schema below:

Key	Description	Type
id	Unique ID	Integer
name	Name of this group	String
valid	Is this group well-formed?	Boolean
installed	... and is it installed?	Boolean
nexthops	Nexthops within this group	Array

Each element within `nexthops` describes a single target within this group, and its structure is described by the JSON below:

Key	Description	Type
nexthop	Name of this nexthop	String
valid	Is this nexthop well-formed?	Boolean

3.16.3 PBR Maps

PBR maps are a way to group policies that we would like to apply to individual interfaces. These policies when applied are matched against incoming packets. If matched the nexthop-group or nexthop is used to forward the packets to the end destination.

`pbr-map NAME seq (1-700)`

Create a pbr-map with NAME and sequence number specified. This command puts you into a new submode for pbr-map specification. To exit this mode type exit or end as per normal conventions for leaving a sub-mode.

`match src-ip PREFIX`

When a incoming packet matches the source prefix specified, take the packet and forward according to the nexthops specified. This command accepts both v4 and v6 prefixes. This command is used in conjunction of the *`match dst-ip PREFIX`* command for matching.

`match dst-ip PREFIX`

When a incoming packet matches the destination prefix specified, take the packet and forward according to the nexthops specified. This command accepts both v4 and v6 prefixes. This command is used in conjunction of the *`match src-ip PREFIX`* command for matching.

`match src-port (1-65535)`

When a incoming packet matches the source port specified, take the packet and forward according to the nexthops specified.

`match dst-port (1-65535)`

When a incoming packet matches the destination port specified, take the packet and forward according to the nexthops specified.

match ip-protocol [tcp|udp]

When a incoming packet matches the specified ip protocol, take the packet and forward according to the nexthops specified.

match mark (1-4294967295)

Select the mark to match. This is a linux only command and if attempted on another platform it will be denied. This mark translates to the underlying *ip rule …. fwmark XXXX* command.

match dscp (DSCP|0-63)

Match packets according to the specified differentiated services code point (DSCP) in the IP header; if this value matches then forward the packet according to the nexthop(s) specified. The passed DSCP value may also be a standard name for a differentiated service code point like cs0 or af11.

You may only specify one dscp per route map sequence; to match on multiple dscp values you will need to create several sequences, one for each value.

match ecn (0-3)

Match packets according to the specified explicit congestion notification (ECN) field in the IP header; if this value matches then forward the packet according to the nexthop(s) specified.

set queue-id (1-65535)

Set the egress port queue identifier for matched packets. The Linux Kernel provider does not currently support packet mangling, so this field will be ignored unless another provider is used.

set pcp (0-7)

Set the 802.1Q priority code point (PCP) for matched packets. A PCP of zero is the defaul (nominally, "best effort"). The Linux Kernel provider does not currently support packet mangling, so this field will be ignored unless another provider is used.

set vlan (1-4094)

Set the VLAN tag for matched packets. Identifiers 0 and 4095 are reserved. The Linux Kernel provider does not currently support packet mangling, so this field will be ignored unless another provider is used.

strip vlan

Strip inner vlan tags from matched packets. The Linux Kernel provider does not currently support packet mangling, so this field will be ignored unless another provider is used. It is invalid to specify both a *strip* and *set vlan* action.

set nexthop-group NAME

Use the nexthop-group NAME as the place to forward packets when the match commands have matched a packet.

set nexthop [A.B.C.D|X:X::X:XX] [interface] [nexthop-vrf NAME]

Use this individual nexthop as the place to forward packets when the match commands have matched a packet.

set vrf unchanged|NAME

If unchanged is set, the rule will use the vrf table the interface is in as its lookup. If NAME is specified, the rule will use that vrf table as its lookup.

Not supported with NETNS VRF backend.

show pbr map [NAME] [detail|json]

Display pbr maps either all or by `NAME`. If `detail` is set, it will give information about the rules unique ID used internally and some extra debugging information about install state for the nexthop/nexthop group. Setting `json` will provide the same information in an array of objects which obey the schema below:

Key	Description	Type
name	Map name	String
valid	Is the map well-formed?	Boolean
policies	Rules to match packets against	Array

Each element of the `policies` array is composed of a handful of objects representing the policies associated with this map. Each policy is described as below (not all fields are required):

Key	Description	Type
id	Unique ID	Integer
sequenceNumber	Order of this policy within the map	Integer
ruleNumber	Rule number to install into	Integer
vrfUnchanged	Use interface's VRF	Boolean
installed	Is this policy installed?	Boolean
installedReason	Why (or why not?)	String
matchSrc	Match packets with this source address	String
matchDst	. . . or with this destination address	String
matchMark	. . . or with this marker	Integer
vrfName	Associated VRF (if relevant)	String
nexthopGroup	This policy's nexthop group (if relevant)	Object

Finally, the `nexthopGroup` object above cotains information we know about the configured nexthop for this policy:

Key	Description	Type
tableId	Nexthop table ID	Integer
name	Name of the nexthop group	String
installed	Is this nexthop group installed?	Boolean
installedInternally	Do we think this group is installed?	Integer

3.16.4 PBR Policy

After you have specified a PBR map, in order for it to be turned on, you must apply the PBR map to an interface. This policy application to an interface causes the policy to be installed into the kernel.

`pbr-policy NAME`

This command is available under interface sub-mode. This turns on the PBR map NAME and allows it to work properly.

Note: This will not dynamically create PBR maps on sub-interfaces (i.e. vlans) even if one is on the master. Each must have the PBR map explicitly added to the interface.

`show pbr interface [NAME] [json]`

Enumerates all interfaces which `pbrd` is keeping track of. Passing `json` will return an array of interfaces; each returned interface will adhere to the JSON schema below:

Key	Description	Type
name	Interface name	String
index	Device Index	Integer
policy	PBR map for this interface	String
valid	Is the map well-formed?	Boolean

`pbr table range (10000-4294966272) (10000-4294966272)`

Set or unset the range used to assign numeric table ID's to new nexthop-group tables. Existing tables will not be modified to fit in this range, so it is recommended to configure this before adding nexthop groups.

See also:

PBR Details

3.16.5 PBR Debugs

`debug pbr events|map|nht|zebra`

Debug pbr in pbrd daemon. You specify what types of debugs to turn on.

3.16.6 PBR Details

Under the covers a PBR map is translated into two separate constructs in the Linux kernel.

The PBR map specified creates a *ip rule …* that is inserted into the Linux kernel that points to a table to use for forwarding once the rule matches.

The creation of a nexthop or nexthop-group is translated to a default route in a table with the nexthops specified as the nexthops for the default route.

3.16.7 Sample configuration

```
nexthop-group TEST
  nexthop 4.5.6.7
  nexthop 5.6.7.8
!
pbr-map BLUE seq 100
  match dst-ip 9.9.9.0/24
  match src-ip 10.10.10.0/24
  set nexthop-group TEST
!
int swp1
  pbr-policy BLUE
```

3.17 RIP

RIP – Routing Information Protocol is widely deployed interior gateway protocol. RIP was developed in the 1970s at Xerox Labs as part of the XNS routing protocol. RIP is a *distance-vector* protocol and is based on the *Bellman-Ford* algorithms. As a distance-vector protocol, RIP router send updates to its neighbors periodically, thus allowing the convergence to a known topology. In each update, the distance to any given network will be broadcast to its neighboring router.

ripd supports RIP version 2 as described in RFC2453 and RIP version 1 as described in RFC1058.

3.17.1 Starting and Stopping ripd

The default configuration file name of *ripd*'s is `ripd.conf`. When invocation *ripd* searches directory /etc/frr. If `ripd.conf` is not there next search current directory.

RIP uses UDP port 520 to send and receive RIP packets. So the user must have the capability to bind the port, generally this means that the user must have superuser privileges. RIP protocol requires interface information maintained by *zebra* daemon. So running *zebra* is mandatory to run *ripd*. Thus minimum sequence for running RIP is like below:

```
# zebra -d
# ripd -d
```

Please note that *zebra* must be invoked before *ripd*.

To stop *ripd*. Please use:

```
kill `cat /var/run/frr/ripd.pid`
```

Certain signals have special meanings to *ripd*.

Signal	Action
`SIGHUP`	Reload configuration file `ripd.conf`. All configurations are reset. All routes learned so far are cleared and removed from routing table.
`SIGUSR1`	Rotate the *ripd* logfile.
`SIGINT` `SIGTERM`	Sweep all installed routes and gracefully terminate.

ripd invocation options. Common options that can be specified (*Common Invocation Options*).

RIP netmask

The netmask features of *ripd* support both version 1 and version 2 of RIP. Version 1 of RIP originally contained no netmask information. In RIP version 1, network classes were originally used to determine the size of the netmask. Class A networks use 8 bits of mask, Class B networks use 16 bits of masks, while Class C networks use 24 bits of mask. Today, the most widely used method of a network mask is assigned to the packet on the basis of the interface that received the packet. Version 2 of RIP supports a variable length subnet mask (VLSM). By extending the subnet mask, the mask can be divided and reused. Each subnet can be used for different purposes such as large to middle size LANs and WAN links. FRR *ripd* does not support the non-sequential netmasks that are included in RIP Version 2.

In a case of similar information with the same prefix and metric, the old information will be suppressed. Ripd does not currently support equal cost multipath routing.

3.17.2 RIP Configuration

`router rip [vrf NAME]`

The *router rip* command is necessary to enable RIP. To disable RIP, use the *no router rip* command. RIP must be enabled before carrying out any of the RIP commands.

`network NETWORK`

Set the RIP enable interface by NETWORK. The interfaces which have addresses matching with NETWORK are enabled.

This group of commands either enables or disables RIP interfaces between certain numbers of a specified network address. For example, if the network for 10.0.0.0/24 is RIP enabled, this would result in all the addresses from 10.0.0.0 to 10.0.0.255 being enabled for RIP. The *no network* command will disable RIP for the specified network.

`network IFNAME`

Set a RIP enabled interface by IFNAME. Both the sending and receiving of RIP packets will be enabled on the port specified in the *network ifname* command. The *no network ifname* command will disable RIP on the specified interface.

`neighbor A.B.C.D`

Specify a RIP neighbor to send updates to. This is required when a neighbor is connected via a network that does not support multicast, or when it is desired to statically define a neighbor. RIP updates will be sent via unicast to each neighbour. Neighbour updates are in addition to any multicast updates sent when an interface is not in passive mode (see the *passive-interface* command). RIP will continue to process updates received from both the neighbor and any received via multicast. The *no neighbor a.b.c.d* command will disable the RIP neighbor.

Below is very simple RIP configuration. Interface *eth0* and interface which address match to *10.0.0.0/8* are RIP enabled.

```
!
router rip
 network 10.0.0.0/8
 network eth0
!
```

`passive-interface (IFNAME|default)`

This command sets the specified interface to passive mode. On passive mode interface, all receiving packets are processed as normal and ripd does not send either multicast or unicast RIP packets except to RIP neighbors specified with *neighbor* command. The interface may be specified as *default* to make ripd default to passive on all interfaces.

The default is to be passive on all interfaces.

`ip split-horizon [poisoned-reverse]`

Control split-horizon on the interface. Default is *ip split-horizon*. If you don't perform split-horizon on the interface, please specify *no ip split-horizon*.

If *poisoned-reverse* is also set, the router sends the poisoned routes with highest metric back to the sending router.

`allow-ecmp [1-MULTIPATH_NUM]`

Control how many ECMP paths RIP can inject for the same prefix. If specified without a number, a maximum is taken (compiled with `--enable-multipath`).

3.17.3 RIP Version Control

RIP can be configured to send either Version 1 or Version 2 packets. The default is to send RIPv2 while accepting both RIPv1 and RIPv2 (and replying with packets of the appropriate version for REQUESTS / triggered updates). The version to receive and send can be specified globally, and further overridden on a per-interface basis if needs be for send and receive separately (see below).

It is important to note that RIPv1 cannot be authenticated. Further, if RIPv1 is enabled then RIP will reply to REQUEST packets, sending the state of its RIP routing table to any remote routers that ask on demand. For a more detailed discussion on the security implications of RIPv1 see *RIP Authentication*.

`version VERSION`

Set RIP version to accept for reads and send. VERSION can be either `1` or `2`.

Disabling RIPv1 by specifying version 2 is STRONGLY encouraged, *RIP Authentication*. This may become the default in a future release.

Default: Send Version 2, and accept either version.

`ip rip send version VERSION`

VERSION can be `1`, `2`, or `1 2`.

This interface command overrides the global rip version setting, and selects which version of RIP to send packets with, for this interface specifically. Choice of RIP Version 1, RIP Version 2, or both versions. In the latter case, where `1 2` is specified, packets will be both broadcast and multicast.

Default: Send packets according to the global version (version 2)

`ip rip receive version VERSION`

VERSION can be `1`, `2`, or `1 2`.

This interface command overrides the global rip version setting, and selects which versions of RIP packets will be accepted on this interface. Choice of RIP Version 1, RIP Version 2, or both.

Default: Accept packets according to the global setting (both 1 and 2).

3.17.4 How to Announce RIP route

`redistribute <babel|bgp|connected|eigrp|isis|kernel|openfabric|ospf|sharp|static|table> [metric (0-16)]`

Redistribute routes from other sources into RIP.

If you want to specify RIP only static routes:

`default-information originate`

`route A.B.C.D/M`

This command is specific to FRR. The *route* command makes a static route only inside RIP. This command should be used only by advanced users who are particularly knowledgeable about the RIP protocol. In most cases, we recommend creating a static route in FRR and redistributing it in RIP using *redistribute static*.

3.17.5 Filtering RIP Routes

RIP routes can be filtered by a distribute-list.

`distribute-list [prefix] LIST <in|out> IFNAME`

You can apply access lists to the interface with a *distribute-list* command. If prefix is specified LIST is a prefix-list. If prefix is not specified then LIST is the access list name. *in* specifies packets being received, and *out* specifies outgoing packets. Finally if an interface is specified it will be applied against a specific interface.

The *distribute-list* command can be used to filter the RIP path. *distribute-list* can apply access-lists to a chosen interface. First, one should specify the access-list. Next, the name of the access-list is used in the distribute-list command. For example, in the following configuration `eth0` will permit only the paths that match the route 10.0.0.0/8

```
!
router rip
 distribute-list private in eth0
!
access-list private permit 10 10.0.0.0/8
access-list private deny any
!
```

distribute-list can be applied to both incoming and outgoing data.

3.17.6 RIP Metric Manipulation

RIP metric is a value for distance for the network. Usually *ripd* increment the metric when the network information is received. Redistributed routes' metric is set to 1.

`default-metric (1-16)`

This command modifies the default metric value for redistributed routes. The default value is 1. This command does not affect connected route even if it is redistributed by *redistribute connected*. To modify connected route's metric value, please use `redistribute connected metric` or *route-map*. *offset-list* also affects connected routes.

`offset-list ACCESS-LIST (in|out)`

`offset-list ACCESS-LIST (in|out) IFNAME`

3.17.7 RIP distance

Distance value is used in zebra daemon. Default RIP distance is 120.

`distance (1-255)`

Set default RIP distance to specified value.

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

Set default RIP distance to specified value when the route's source IP address matches the specified prefix.

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

Set default RIP distance to specified value when the route's source IP address matches the specified prefix and the specified access-list.

3.17.8 RIP route-map

Usage of *ripd*'s route-map support.

Optional argument route-map MAP_NAME can be added to each *redistribute* statement.

```
redistribute static [route-map MAP_NAME]
redistribute connected [route-map MAP_NAME]
.....
```

Cisco applies route-map _before_ routes will exported to rip route table. In current FRR's test implementation, *ripd* applies route-map after routes are listed in the route table and before routes will be announced to an interface (something like output filter). I think it is not so clear, but it is draft and it may be changed at future.

Route-map statement (*Route Maps*) is needed to use route-map functionality.

`match interface WORD`

This command match to incoming interface. Notation of this match is different from Cisco. Cisco uses a list of interfaces - NAME1 NAME2 … NAMEN. Ripd allows only one name (maybe will change in the future). Next - Cisco means interface which includes next-hop of routes (it is somewhat similar to "ip next-hop" statement). Ripd means interface where this route will be sent. This difference is because "next-hop" of same routes which sends to different interfaces must be different. Maybe it'd be better to made new matches - say "match interface-out NAME" or something like that.

`match ip address WORD`

match ip address prefix-list WORD
Match if route destination is permitted by access-list.

match ip next-hop WORD

match ip next-hop prefix-list WORD
Match if route next-hop (meaning next-hop listed in the rip route-table as displayed by "show ip rip") is permitted by access-list.

match metric (0-4294967295)
This command match to the metric value of RIP updates. For other protocol compatibility metric range is shown as (0-4294967295). But for RIP protocol only the value range (0-16) make sense.

set ip next-hop A.B.C.D
This command set next hop value in RIPv2 protocol. This command does not affect RIPv1 because there is no next hop field in the packet.

set metric (0-4294967295)
Set a metric for matched route when sending announcement. The metric value range is very large for compatibility with other protocols. For RIP, valid metric values are from 1 to 16.

3.17.9 RIP Authentication

RIPv2 allows packets to be authenticated via either an insecure plain text password, included with the packet, or via a more secure MD5 based HMAC (keyed-Hashing for Message AuthentiCation), RIPv1 can not be authenticated at all, thus when authentication is configured *ripd* will discard routing updates received via RIPv1 packets.

However, unless RIPv1 reception is disabled entirely, *RIP Version Control*, RIPv1 REQUEST packets which are received, which query the router for routing information, will still be honoured by *ripd*, and *ripd* WILL reply to such packets. This allows *ripd* to honour such REQUESTs (which sometimes is used by old equipment and very simple devices to bootstrap their default route), while still providing security for route updates which are received.

In short: Enabling authentication prevents routes being updated by unauthenticated remote routers, but still can allow routes (I.e. the entire RIP routing table) to be queried remotely, potentially by anyone on the internet, via RIPv1.

To prevent such unauthenticated querying of routes disable RIPv1, *RIP Version Control*.

ip rip authentication mode md5
Set the interface with RIPv2 MD5 authentication.

ip rip authentication mode text
Set the interface with RIPv2 simple password authentication.

ip rip authentication string STRING
RIP version 2 has simple text authentication. This command sets authentication string. The string must be shorter than 16 characters.

ip rip authentication key-chain KEY-CHAIN
Specify Keyed MD5 chain.

```
!
key chain test
 key 1
  key-string test
!
interface eth1
 ip rip authentication mode md5
 ip rip authentication key-chain test
!
```

3.17.10 RIP Timers

timers basic UPDATE TIMEOUT GARBAGE

RIP protocol has several timers. User can configure those timers' values by *timers basic* command.

The default settings for the timers are as follows:

- The update timer is 30 seconds. Every update timer seconds, the RIP process is awakened to send an unsolicited Response message containing the complete routing table to all neighboring RIP routers.
- The timeout timer is 180 seconds. Upon expiration of the timeout, the route is no longer valid; however, it is retained in the routing table for a short time so that neighbors can be notified that the route has been dropped.
- The garbage collect timer is 120 seconds. Upon expiration of the garbage-collection timer, the route is finally removed from the routing table.

The `timers basic` command allows the the default values of the timers listed above to be changed.

3.17.11 Show RIP Information

To display RIP routes.

show ip rip [vrf NAME]

Show RIP routes.

The command displays all RIP routes. For routes that are received through RIP, this command will display the time the packet was sent and the tag information. This command will also display this information for routes redistributed into RIP.

show ip rip [vrf NAME] status

The command displays current RIP status. It includes RIP timer, filtering, version, RIP enabled interface and RIP peer information.

```
ripd> **show ip rip status**
Routing Protocol is "rip"
  Sending updates every 30 seconds with +/-50%, next due in 35 seconds
  Timeout after 180 seconds, garbage collect after 120 seconds
  Outgoing update filter list for all interface is not set
  Incoming update filter list for all interface is not set
  Default redistribution metric is 1
  Redistributing: kernel connected
  Default version control: send version 2, receive version 2
    Interface  Send  Recv
  Routing for Networks:
    eth0
    eth1
    1.1.1.1
    203.181.89.241
  Routing Information Sources:
    Gateway    BadPackets BadRoutes  Distance Last Update
```

3.17.12 RIP Debug Commands

Debug for RIP protocol.

`debug rip events`
Shows RIP events. Sending and receiving packets, timers, and changes in interfaces are events shown with *ripd*.

`debug rip packet`
Shows display detailed information about the RIP packets. The origin and port number of the packet as well as a packet dump is shown.

`debug rip zebra`
This command will show the communication between *ripd* and *zebra*. The main information will include addition and deletion of paths to the kernel and the sending and receiving of interface information.

`show debugging rip`
Shows all information currently set for ripd debug.

3.17.13 Sample configuration

```
debug rip events
debug rip packet

router rip
 network 11.0.0.0/8
 network eth0
 route 10.0.0.0/8
 distribute-list private-only in eth0

access-list private-only permit 10.0.0.0/8
access-list private-only deny any
```

3.18 RIPng

ripngd supports the RIPng protocol as described in **RFC 2080**. It's an IPv6 reincarnation of the RIP protocol.

3.18.1 Invoking ripngd

There are no *ripngd* specific invocation options. Common options can be specified (*Common Invocation Options*).

3.18.2 ripngd Configuration

Currently ripngd supports the following commands:

`router ripng [vrf NAME]`
Enable RIPng.

`network NETWORK`
Set RIPng enabled interface by NETWORK.

`network IFNAME`
Set RIPng enabled interface by IFNAME.

route NETWORK
Set RIPng static routing announcement of NETWORK.

allow-ecmp [1-MULTIPATH_NUM]
Control how many ECMP paths RIPng can inject for the same prefix. If specified without a number, a maximum is taken (compiled with `--enable-multipath`).

3.18.3 ripngd Terminal Mode Commands

show ipv6 ripng [vrf NAME] status

show debugging ripng

debug ripng events

debug ripng packet

debug ripng zebra

3.18.4 ripngd Filtering Commands

RIPng routes can be filtered by a distribute-list.

distribute-list [prefix] LIST <in|out> IFNAME
You can apply access lists to the interface with a *distribute-list* command. If prefix is specified LIST is a prefix-list. If prefix is not specified then LIST is the access list name. *in* specifies packets being received, and *out* specifies outgoing packets. Finally if an interface is specified it will be applied against a specific interface.

The `distribute-list` command can be used to filter the RIPNG path. `distribute-list` can apply access-lists to a chosen interface. First, one should specify the access-list. Next, the name of the access-list is used in the distribute-list command. For example, in the following configuration `eth0` will permit only the paths that match the route 10.0.0.0/8

```
!
router ripng
 distribute-list private in eth0
!
access-list private permit 10 10.0.0.0/8
access-list private deny any
!
```

distribute-list can be applied to both incoming and outgoing data.

3.18.5 Sample configuration

```
debug ripng events
debug ripng packet

router ripng
 network sit1
 route 3ffe:506::0/32
 distribute-list local-only out sit1

```

(continues on next page)

(continued from previous page)

```
ipv6 access-list local-only permit 3ffe:506::0/32
ipv6 access-list local-only deny any
```

3.19 SHARP

SHARP (Super Happy Advanced Routing Process) is a daemon that provides miscellaneous functionality used for testing FRR and creating proof-of-concept labs.

3.19.1 Starting SHARP

Default configuration file for *sharpd* is `sharpd.conf`. The typical location of `sharpd.conf` is /etc/frr/sharpd.conf.

If the user is using integrated config, then `sharpd.conf` need not be present and the `frr.conf` is read instead.

SHARP supports all the common FRR daemon start options which are documented elsewhere.

3.19.2 Using SHARP

All sharp commands are under the enable node and preceded by the `sharp` keyword. At present, no sharp commands will be preserved in the config.

`sharp install routes A.B.C.D <nexthop <E.F.G.H|X:X::X:X>|nexthop-group NAME> (1-1000000) [instance (0-255)] [repeat (2-1000)] [opaque WORD]`

Install up to 1,000,000 (one million) /32 routes starting at `A.B.C.D` with specified nexthop `E.F.G.H` or `X:X::X:X`. The nexthop is a `NEXTHOP_TYPE_IPV4` or `NEXTHOP_TYPE_IPV6` and must be reachable to be installed into the kernel. Alternatively a nexthop-group NAME can be specified and used as the nexthops. The routes are installed into zebra as `ZEBRA_ROUTE_SHARP` and can be used as part of a normal route redistribution. Route installation time is noted in the debug log. When zebra successfully installs a route into the kernel and SHARP receives success notifications for all routes this is logged as well. Instance (0-255) if specified causes the routes to be installed in a different instance. If repeat is used then we will install/uninstall the routes the number of times specified. If the keyword opaque is specified then the next word is sent down to zebra as part of the route installation.

`sharp remove routes A.B.C.D (1-1000000)`

Remove up to 1,000,000 (one million) /32 routes starting at `A.B.C.D`. The routes are removed from zebra. Route deletion start is noted in the debug log and when all routes have been successfully deleted the debug log will be updated with this information as well.

`sharp data route`

Allow end user doing route install and deletion to get timing information from the vty or vtysh instead of having to read the log file. This command is informational only and you should look at sharp_vty.c for explanation of the output as that it may change.

`sharp label <ipv4|ipv6> vrf NAME label (0-1000000)`

Install a label into the kernel that causes the specified vrf NAME table to be used for pop and forward operations when the specified label is seen.

`sharp watch <nexthop <A.B.C.D|X:X::X:X>|import <A.B.C.D/M:X:X::X:X/M> [connected]`

Instruct zebra to monitor and notify sharp when the specified nexthop is changed. The notification from zebra is written into the debug log. The nexthop or import choice chooses the type of nexthop we are asking zebra to watch for us. This choice affects zebra's decision on what matches. Connected tells zebra whether or not that we want the route matched against to be a static or connected route for the nexthop keyword, for the import keyword connected means exact match. The no form of the command obviously turns this watching off.

`sharp data nexthop`
: Allow end user to dump associated data with the nexthop tracking that may have been turned on.

`sharp watch [vrf NAME] redistribute ROUTETYPE`
: Allow end user to monitor redistributed routes of ROUTETYPE origin.

`sharp lsp [update] (0-100000) nexthop-group NAME [prefix A.B.C.D/M TYPE [instance (0-255)]]`
: Install an LSP using the specified in-label, with nexthops as listed in nexthop-group `NAME`. If `update` is included, the update path is used. The LSP is installed as type ZEBRA_LSP_SHARP. If `prefix` is specified, an existing route with type `TYPE` (and optional `instance` id) will be updated to use the LSP.

`sharp remove lsp (0-100000) nexthop-group NAME [prefix A.B.C.D/M TYPE [instance (0-255)]]`
: Remove a SHARPD LSP that uses the specified in-label, where the nexthops are specified in nexthop-group `NAME`. If `prefix` is specified, remove label bindings from the route of type `TYPE` also.

`sharp send opaque type (1-255) (1-1000)`
: Send opaque ZAPI messages with subtype `type`. Sharpd will send a stream of messages if the count is greater than one.

`sharp send opaque unicast type (1-255) PROTOCOL [{instance (0-1000) | session (1-1000)}] (1-1000)`
: Send unicast opaque ZAPI messages with subtype `type`. The protocol, instance, and session_id identify a single target zapi client. Sharpd will send a stream of messages if the count is greater than one.

`sharp send opaque <reg | unreg> PROTOCOL [{instance (0-1000) | session (1-1000)}] type (1-1000)`
: Send opaque ZAPI registration and unregistration messages for a single subtype. The messages must specify a protocol daemon by name, and can include optional zapi `instance` and `session` values.

`sharp create session (1-1024)`
: Create an additional zapi client session for testing, using the specified session id.

`sharp remove session (1-1024)`
: Remove a test zapi client session that was created with the specified session id.

`sharp neigh discover [vrf NAME] <A.B.C.D|X:X::X:X> IFNAME`
: Send an ARP/NDP request to trigger the addition of a neighbor in the ARP table.

`sharp import-te`
: Import Traffic Engineering Database produced by OSPF or IS-IS.

`show sharp ted [verbose|json]`

`show sharp ted [<vertex [A.B.C.D]|edge [A.B.C.D]|subnet [A.B.C.D/M]>] [verbose|json]`
: Show imported Traffic Engineering Data Base

`show sharp cspf source <A.B.C.D|X:X:X:X> destination <A.B.C.D|X:X:X:X> <metric|te-metric|delay> (0-16777215) [rsv-bw (0-7) BANDWIDTH]`
: Show the result of a call to the Constraint Shortest Path First (CSPF) algorithm that allows to compute a path between a source and a destination under various constraints. Standard Metric, TE Metric, Delay and Bandwidth are supported constraints. Prior to use this function, it is necessary to import a Traffic Engineering Database with *sharp import-te* command (see above).

`sharp install seg6-routes [vrf NAME] <A.B.C.D|X:X::X:X> nexthop-seg6 X:X::X:X encap X:X::X:X (1-1000000)`
: This command installs a route for SRv6 Transit behavior (on Linux it is known as seg6 route). The count, destination, vrf, etc. have the same meaning as in the `sharp install routes` command. With this command, sharpd will request zebra to configure seg6 route via ZEBRA_ROUTE_ADD ZAPI. As in the following example.

```
router# sharp install seg6-routes 1::A nexthop-seg6 2001::2 encap A:: 1
router# sharp install seg6-routes 1::B nexthop-seg6 2001::2 encap B:: 1

router# show ipv6 route
D>* 1::A/128 [150/0] via 2001::2, dum0, seg6 a::, weight 1, 00:00:01
D>* 1::B/128 [150/0] via 2001::2, dum0, seg6 b::, weight 1, 00:00:01

bash# ip -6 route list
1::A  encap seg6 mode encap segs 1 [ a:: ] via 2001::2 dev dum0 proto 194 metric 20 pref␣
↪medium
1::B  encap seg6 mode encap segs 1 [ b:: ] via 2001::2 dev dum0 proto 194 metric 20 pref␣
↪medium
```

sharp install seg6local-routes [vrf NAME] X:X::X:X nexthop-seg6local NAME ACTION ARGS.. (1-1000000)

This command installs a route for SRv6 Endpoint behavior (on Linux it is known as seg6local route). The count, destination, vrf, etc. have the same meaning as in the `sharp install routes` command. With this command, sharpd will request zebra to configure seg6local route via ZEBRA_ROUTE_ADD ZAPI. As in the following example.

There are many End Functions defined in SRv6, which have been standardized in RFC 8986. The current implementation supports End, End.X, End.T, End.DX4, End.DT6 and End.DT46, which can be configured as follows.

```
router# sharp install seg6local-routes 1::1 nexthop-seg6local dum0 End 1
router# sharp install seg6local-routes 1::2 nexthop-seg6local dum0 End_X 2001::1 1
router# sharp install seg6local-routes 1::3 nexthop-seg6local dum0 End_T 10 1
router# sharp install seg6local-routes 1::4 nexthop-seg6local dum0 End_DX4 10.0.0.1 1
router# sharp install seg6local-routes 1::5 nexthop-seg6local dum0 End_DT6 10 1
router# sharp install seg6local-routes 1::6 nexthop-seg6local dum0 End_DT46 10 1

router# show ipv6 route
D>* 1::1/128 [150/0] is directly connected, dum0, seg6local End USP, weight 1, 00:00:05
D>* 1::2/128 [150/0] is directly connected, dum0, seg6local End.X nh6 2001::1, weight 1,␣
↪00:00:05
D>* 1::3/128 [150/0] is directly connected, dum0, seg6local End.T table 10, weight 1,␣
↪00:00:05
D>* 1::4/128 [150/0] is directly connected, dum0, seg6local End.DX4 nh4 10.0.0.1, weight␣
↪1, 00:00:05
D>* 1::5/128 [150/0] is directly connected, dum0, seg6local End.DT6 table 10, weight 1,␣
↪00:00:05
D>* 1::6/128 [150/0] is directly connected, dum0, seg6local End.DT46 table 10, weight 1,␣
↪00:00:05

bash# ip -6 route
1::1  encap seg6local action End dev dum0 proto 194 metric 20 pref medium
1::2  encap seg6local action End.X nh6 2001::1 dev dum0 proto 194 metric 20 pref medium
1::3  encap seg6local action End.T table 10 dev dum0 proto 194 metric 20 pref medium
1::4  encap seg6local action End.DX4 nh4 10.0.0.1 dev dum0 proto 194 metric 20 pref␣
↪medium
1::5  encap seg6local action End.DT6 table 10 dev dum0 proto 194 metric 20 pref medium
1::6  encap seg6local action End.DT46 table 10 dev dum0 proto 194 metric 20 pref medium
```

show sharp segment-routing srv6

This command shows us what SRv6 locator chunk, sharp is holding as zclient. An SRv6 locator is defined for each SRv6 router, and a single locator may be shared by multiple protocols.

In the FRRouting implementation, the Locator chunk get request is executed by a routing protocol daemon such as sharpd or bgpd, And then Zebra allocates a Locator Chunk, which is a subset of the Locator Prefix, and notifies the requesting protocol daemon of this information.

This command example shows how the locator chunk of sharpd itself is allocated.

```
router# show segment-routing srv6 locator
Locator:
Name                 ID            2 2001:db8:2:2::/64        Up

router# show sharp segment-routing srv6
Locator loc1 has 1 prefix chunks
  2001:db8:1:1::/64
```

`sharp srv6-manager get-locator-chunk`

This command requests the SRv6 locator to allocate a locator chunk via ZAPI. This chunk can be owned by the protocol daemon, and the chunk obtained by sharpd will not be used by the SRv6 mechanism of another routing protocol.

Since this request is made asynchronously, it can be issued before the SRv6 locator is configured on the zebra side, and as soon as it is ready on the zebra side, sharpd can check the allocated locator chunk via zapi.

```
router# show segment-routing srv6 locator loc1 detail
Name: loc1
Prefix: 2001:db8:1:1::/64
Chunks:
- prefix: 2001:db8:1:1::/64, owner: system

router# show sharp segment-routing srv6
(nothing)

router# sharp srv6-manager get-locator-chunk loc1

router# show segment-routing srv6 locator loc1 detail
Name: loc1
Prefix: 2001:db8:1:1::/64
Chunks:
- prefix: 2001:db8:1:1::/64, owner: sharp

router# show sharp segment-routing srv6
Locator loc1 has 1 prefix chunks
  2001:db8:1:1::/64
```

`sharp srv6-manager release-locator-chunk`

This command releases a locator chunk that has already been allocated by ZAPI. The freed chunk will have its owner returned to the system and will be available to another protocol daemon.

```
router# show segment-routing srv6 locator loc1 detail
Name: loc1
Prefix: 2001:db8:1:1::/64
Chunks:
- prefix: 2001:db8:1:1::/64, owner: sharp

```

(continues on next page)

(continued from previous page)

```
router# show sharp segment-routing srv6
Locator loc1 has 1 prefix chunks
  2001:db8:1:1::/64

router# sharp srv6-manager release-locator-chunk loc1

router# show segment-routing srv6 locator loc1 detail
Name: loc1
Prefix: 2001:db8:1:1::/64
Chunks:
- prefix: 2001:db8:1:1::/64, owner: system

router# show sharp segment-routing srv6
(nothing)
```

sharp interface IFNAME protodown
Set an interface protodown.

3.20 STATIC

STATIC is a daemon that handles the installation and deletion of static routes.

3.20.1 Starting STATIC

Default configuration file for *staticd* is `staticd.conf`. The typical location of `staticd.conf` is /etc/frr/staticd.conf.

If the user is using integrated config, then `staticd.conf` need not be present and the `frr.conf` is read instead.

If the user has not fully upgraded to using the staticd.conf and still has a non-integrated config with zebra.conf holding the static routes, *staticd* will read in the `zebrad.conf` as a backup.

STATIC supports all the common FRR daemon start options which are documented elsewhere.

3.20.2 Static Route Commands

Static routing is a very fundamental feature of routing technology. It defines a static prefix and gateway, with several possible forms.

ip route NETWORK GATEWAY [DISTANCE] [table TABLENO] [nexthop-vrf VRFNAME] [vrf VRFNAME]

ip route NETWORK IFNAME [DISTANCE] [table TABLENO] [nexthop-vrf VRFNAME] [vrf VRFNAME]

ip route NETWORK GATEWAY IFNAME [DISTANCE] [onlink] [table TABLENO] [nexthop-vrf VRFNAME] [vrf VRFNAME]

ip route NETWORK (Null0|blackhole|reject) [DISTANCE] [table TABLENO] [nexthop-vrf VRFNAME] [vrf VRFNAME

ipv6 route NETWORK [from SRCPREFIX] GATEWAY [DISTANCE] [table TABLENO] [nexthop-vrf VRFNAME] [vrf VRFNA

ipv6 route NETWORK [from SRCPREFIX] IFNAME [DISTANCE] [table TABLENO] [nexthop-vrf VRFNAME] [vrf VRFNAM

ipv6 route NETWORK [from SRCPREFIX] GATEWAY IFNAME [DISTANCE] [onlink] [table TABLENO] [nexthop-vrf VRF

ipv6 route NETWORK [from SRCPREFIX] (Null0|blackhole|reject) [DISTANCE] [table TABLENO] [nexthop-vrf VR
NETWORK is destination prefix with a valid v4 or v6 network based upon initial form of the command.

GATEWAY is the IP address to use as next-hop for the prefix. Currently, it must match the v4 or v6 route type specified at the start of the command.

IFNAME is the name of the interface to use as next-hop. If only IFNAME is specified (without GATEWAY), a connected route will be created.

When both IFNAME and GATEWAY are specified together, it binds the route to the specified interface. In this case, it is also possible to specify `onlink` to force the kernel to consider the next-hop as "on link" on the given interface.

Alternatively, the gateway can be specified as `Null0` or `blackhole` to create a blackhole route that drops all traffic. It can also be specified as `reject` to create an unreachable route that rejects traffic with ICMP "Destination Unreachable" messages.

TABLENO is an optional parameter for namespaces that allows you to create the route in a specified table associated with the vrf namespace. `table` will be rejected if you are not using namespace based vrfs.

`vrf` VRFNAME allows you to create the route in a specified vrf.

`nexthop-vrf` VRFNAME allows you to create a leaked route with a nexthop in the specified VRFNAME. `nexthop-vrf` cannot be currently used with namespace based vrfs.

The IPv6 variant allows the installation of a static source-specific route with the SRCPREFIX sub command. These routes are currently supported on Linux operating systems only, and perform AND matching on packet's destination and source addresses in the kernel's forwarding path. Note that destination longest-prefix match is "more important" than source LPM, e.g. `2001:db8:1::/64 from 2001:db8::/48` will win over `2001:db8::/48 from 2001:db8:1::/64` if both match.

3.20.3 Multiple nexthop static route

To create multiple nexthops to the same NETWORK (also known as a multipath route), just reenter the same network statement with different nexthop information.

```
ip route 10.0.0.1/32 10.0.0.2
ip route 10.0.0.1/32 10.0.0.3
ip route 10.0.0.1/32 eth0
```

If there is no route to 10.0.0.2 and 10.0.0.3, and interface eth0 is reachable, then the last route is installed into the kernel.

If zebra has been compiled with multipath support, and both 10.0.0.2 and 10.0.0.3 are reachable, zebra will install a multipath route via both nexthops, if the platform supports this.

```
router> show ip route
S>  10.0.0.1/32 [1/0] via 10.0.0.2 inactive
    via 10.0.0.3 inactive
  *       is directly connected, eth0
```

```
ip route 10.0.0.0/8 10.0.0.2
ip route 10.0.0.0/8 10.0.0.3
ip route 10.0.0.0/8 null0 255
```

This will install a multipath route via the specified next-hops if they are reachable, as well as a high-distance blackhole route, which can be useful to prevent traffic destined for a prefix to match less-specific routes (e.g. default) should the specified gateways not be reachable. E.g.:

```
router> show ip route 10.0.0.0/8
Routing entry for 10.0.0.0/8
  Known via "static", distance 1, metric 0
    10.0.0.2 inactive
    10.0.0.3 inactive

Routing entry for 10.0.0.0/8
  Known via "static", distance 255, metric 0
    directly connected, Null0
```

Also, if the user wants to configure a static route for a specific VRF, then a specific VRF configuration mode is available. After entering into that mode with *vrf VRF* the user can enter the same route command as before, but this time, the route command will apply to the VRF.

```
# case with VRF
configure
vrf r1-cust1
 ip route 10.0.0.0/24 10.0.0.2
exit-vrf
```

3.20.4 SR-TE Route Commands

It is possible to specify a route using a SR-TE policy configured in Zebra.

e.g. to use the SR-TE policy with endpoint 6.6.6.6 and color 123 to reach the network 9.9.9.9/24:

```
ip route 9.9.9.9/24 6.6.6.6 color 123
```

3.21 VNC and VNC-GW

This chapter describes how to use VNC (Virtual Network Control) services, including NVA (Network Virtualization Authority) and VNC-GW (VNC Gateway) functions. Background information on NVAs, NVE (Network Virtualization Edge) s, UN (Underlay Network) s, and VN (Virtual Network) is available from the IETF. VNC-GW s support the import/export of routing information between VNC and CE (customer edge) routers operating within a VN. Both IP/Layer 3 (L3) VNs, and IP with Ethernet/Layer 2 (L2) VNs are supported.

BGP, with IP VPNs and Tunnel Encapsulation, is used to distribute VN information between NVAs. BGP based IP VPN support is defined in **RFC 4364**, and **RFC 4659**. Encapsulation information is provided via the Tunnel Encapsulation Attribute, **RFC 5512**.

The protocol that is used to communicate routing and Ethernet / Layer 2 (L2) forwarding information between NVAs and NVEs is referred to as the Remote Forwarder Protocol (RFP). *OpenFlow* is an example RFP. Specific RFP implementations may choose to implement either a *hard-state* or *soft-state* prefix and address registration model. To support a *soft-state* refresh model, a *lifetime* in seconds is associated with all registrations and responses.

The chapter also provides sample configurations for basic example scenarios.

3.21.1 Configuring VNC

Virtual Network Control (VNC) service configuration commands appear in the *router bgp* section of the BGPD configuration file (*Miscellaneous Configuration Examples*). The commands are broken down into the following areas:

- *General VNC* configuration applies to general VNC operation and is primarily used to control the method used to advertise tunnel information.
- *Remote Forwarder Protocol (RFP)* configuration relates to the protocol used between NVAs and NVEs.
- *VNC Defaults* provides default parameters for registered NVEs.
- *VNC NVE Group* provides for configuration of a specific set of registered NVEs and overrides default parameters.
- *Redistribution* and *Export* control VNC-GW operation, i.e., the import/export of routing information between VNC and customer edge routers (CE s) operating within a VN.

General VNC Configuration

RFP Related Configuration

The protocol that is used to communicate routing and Ethernet / L2 forwarding information between NVAs and NVEs is referred to as the Remote Forwarder Protocol (RFP). Currently, only a simple example RFP is included in FRR. Developers may use this example as a starting point to integrate FRR with an RFP of their choosing, e.g., *OpenFlow*. The example code includes the following sample configuration:

`rfp example-config-value VALUE`

This is a simple example configuration parameter included as part of the RFP example code. VALUE must be in the range of 0 to 4294967295.

VNC Defaults Configuration

The VNC Defaults section allows the user to specify default values for configuration parameters for all registered NVEs. Default values are overridden by *VNC NVE Group Configuration*.

`vnc defaults`

Enter VNC configuration mode for specifying VNC default behaviors. Use *exit-vnc* to leave VNC configuration mode. *vnc defaults* is optional.

```
vnc defaults
... various VNC defaults
exit-vnc
```

These are the statements that can appear between `vnc defaults` and `exit-vnc`. Documentation for these statements is given in *VNC NVE Group Configuration*.

- *`rt import RT-LIST`*
- *`rt export RT-LIST`*
- *`rt both RT-LIST`*
- *`rd ROUTE-DISTINGUISHER`*
- *`l2rd NVE-ID-VALUE`*
- *`response-lifetime LIFETIME|infinite`*
- *`export bgp|zebra route-map MAP-NAME`*

- *export bgp|zebra no route-map*

exit-vnc
Exit VNC configuration mode.

VNC NVE Group Configuration

A NVE Group corresponds to a specific set of NVEs. A Client NVE is assigned to an NVE Group based on whether there is a match for either its virtual or underlay network address against the VN and/or UN address prefixes specified in the NVE Group definition. When an NVE Group definition specifies both VN and UN address prefixes, then an NVE must match both prefixes in order to be assigned to the NVE Group. In the event that multiple NVE Groups match based on VN and/or UN addresses, the NVE is assigned to the first NVE Group listed in the configuration. If an NVE is not assigned to an NVE Group, its messages will be ignored.

Configuration values specified for an NVE group apply to all member NVEs and override configuration values specified in the VNC Defaults section.

At least one `nve-group` is mandatory for useful VNC operation.

vnc nve-group NAME
Enter VNC configuration mode for defining the NVE group *name*. Use *exit* or *exit-vnc* to exit group configuration mode.

```
vnc nve-group group1
... configuration commands
exit-vnc
```

The following statements are valid in an NVE group definition:

l2rd NVE-ID-VALUE
Set the value used to distinguish NVEs connected to the same physical Ethernet segment (i.e., at the same location)[1].

The nve-id subfield may be specified as either a literal value in the range 1-255, or it may be specified as *auto:vn*, which means to use the least-significant octet of the originating NVE's VN address.

prefix vn|un A.B.C.D/M|X:X::X:X/M
Specify the matching prefix for this NVE group by either virtual-network address (*vn*) or underlay-network address (*un*). Either or both virtual-network and underlay-network prefixes may be specified. Subsequent virtual-network or underlay-network values within a *vnc nve-group exit-vnc* block override their respective previous values.

These prefixes are used only for determining assignments of NVEs to NVE Groups.

rd ROUTE-DISTINGUISHER
Specify the route distinguisher for routes advertised via BGP VPNs. The route distinguisher must be in one of these forms:

- `IPv4-address:two-byte-integer`
- `four-byte-autonomous-system-number:two-byte-integer`
- `two-byte-autonomous-system-number:four-byte-integer`
- `auto:vn:two-byte-integer`

Routes originated by NVEs in the NVE group will use the group's specified *route-distinguisher* when they are advertised via BGP. If the *auto* form is specified, it means that a matching NVE has its

[1] The nve-id is carried in the route distinguisher. It is the second octet of the eight-octet route distinguisher generated for Ethernet / L2 advertisements. The first octet is a constant 0xFF, and the third through eighth octets are set to the L2 ethernet address being advertised.

RD set to `rd_type=IP=1:IPv4-address=VN-address:two-byte-integer`, for IPv4 VN addresses and `rd_type=IP=1:IPv4-address=Last-four-bytes-of-VN-address:two-byte-integer`, for IPv6 VN addresses.

If the NVE group definition does not specify a *route-distinguisher*, then the default *route-distinguisher* is used. If neither a group nor a default *route-distinguisher* is configured, then the advertised RD is set to `two-byte-autonomous-system-number=0:four-byte-integer=0`.

`response-lifetime LIFETIME|infinite`

Specify the response lifetime, in seconds, to be included in RFP response messages sent to NVEs. If the value 'infinite' is given, an infinite lifetime will be used.

Note that this parameter is not the same as the lifetime supplied by NVEs in RFP registration messages. This parameter does not affect the lifetime value attached to routes sent by this server via BGP.

If the NVE group definition does not specify a *response-lifetime*, the default *response-lifetime* will be used. If neither a group nor a default *response-lifetime* is configured, the value 3600 will be used. The maximum response lifetime is 2147483647.

`rt export RT-LIST`

`rt import RT-LIST`

`rt both RT-LIST`

Specify route target import and export lists. *rt-list* is a space-separated list of route targets, each element of which is in one of the following forms:

- `IPv4-address:two-byte-integer`
- `four-byte-autonomous-system-number:two-byte-integer`
- `two-byte-autonomous-system-number:four-byte-integer`

The first form, *rt export*, specifies an *export rt-list*. The *export rt-list* will be attached to routes originated by NVEs in the NVE group when they are advertised via BGP. If the NVE group definition does not specify an *export rt-list*, then the default *export rt-list* is used. If neither a group nor a default *export rt-list* is configured, then no RT list will be sent; in turn, these routes will probably not be processed by receiving NVAs.

The second form, *rt import* specifies an *import rt-list*, which is a filter for incoming routes. In order to be made available to NVEs in the group, incoming BGP VPN routes must have RT lists that have at least one route target in common with the group's *import rt-list*.

If the NVE group definition does not specify an import filter, then the default *import rt-list* is used. If neither a group nor a default *import rt-list* is configured, there can be no RT intersections when receiving BGP routes and therefore no incoming BGP routes will be processed for the group.

The third, *rt both*, is a shorthand way of specifying both lists simultaneously, and is equivalent to *rt export `rt-list`* followed by *rt import `rt-list`*.

`export bgp|zebra route-map MAP-NAME`

Specify that the named route-map should be applied to routes being exported to bgp or zebra. This parameter is used in conjunction with *Configuring Export of Routes to Other Routing Protocols*. This item is optional.

`export bgp|zebra no route-map`

Specify that no route-map should be applied to routes being exported to bgp or zebra. This parameter is used in conjunction with *Configuring Export of Routes to Other Routing Protocols*. This item is optional.

`export bgp|zebra ipv4|ipv6 prefix-list LIST-NAME`

Specify that the named prefix-list filter should be applied to routes being exported to bgp or zebra. Prefix-lists for ipv4 and ipv6 are independent of each other. This parameter is used in conjunction with *Configuring Export of Routes to Other Routing Protocols*. This item is optional.

export bgp|zebra no ipv4|ipv6 prefix-list

Specify that no prefix-list filter should be applied to routes being exported to bgp or zebra. This parameter is used in conjunction with *Configuring Export of Routes to Other Routing Protocols*. This item is optional.

VNC L2 Group Configuration

The route targets advertised with prefixes and addresses registered by an NVE are determined based on the NVE's associated VNC NVE Group Configuration, *VNC NVE Group Configuration*. Layer 2 (L2) Groups are used to override the route targets for an NVE's Ethernet registrations based on the Logical Network Identifier and label value. A Logical Network Identifier is used to uniquely identify a logical Ethernet segment and is conceptually similar to the Ethernet Segment Identifier defined in **RFC 7432**. Both the Logical Network Identifier and Label are passed to VNC via RFP prefix and address registration.

Note that a corresponding NVE group configuration must be present, and that other NVE associated configuration information, notably RD, is not impacted by L2 Group Configuration.

vnc l2-group NAME

Enter VNC configuration mode for defining the L2 group *name*. Use *exit* or *exit-vnc* to exit group configuration mode.

```
vnc l2-group group1
  ... configuration commands
exit-vnc
```

Delete the L2 group named *name*.

The following statements are valid in a L2 group definition:

logical-network-id VALUE

Define the Logical Network Identifier with a value in the range of 0-4294967295 that identifies the logical Ethernet segment.

labels LABEL-LIST

Add or remove labels associated with the group. *label-list* is a space separated list of label values in the range of 0-1048575.

rt import RT-TARGET

rt export RT-TARGET

rt both RT-TARGET

Specify the route target import and export value associated with the group. A complete definition of these parameters is given above, *VNC NVE Group Configuration*.

Configuring Redistribution of Routes from Other Routing Protocols

Routes from other protocols (including BGP) can be provided to VNC (both for RFP and for redistribution via BGP) from three sources: the zebra kernel routing process; directly from the main (default) unicast BGP RIB; or directly from a designated BGP unicast exterior routing RIB instance.

The protocol named in the *vnc redistribute* command indicates the route source: *bgp-direct* routes come directly from the main (default) unicast BGP RIB and are available for RFP and are redistributed via BGP; *bgp-direct-to-nve-groups* routes come directly from a designated BGP unicast routing RIB and are made available only to RFP; and routes from other protocols come from the zebra kernel routing process. Note that the zebra process does not need to be active if only *bgp-direct* or *bgp-direct-to-nve-groups* routes are used.

zebra routes

Routes originating from protocols other than BGP must be obtained via the zebra routing process. Redistribution of these routes into VNC does not support policy mechanisms such as prefix-lists or route-maps.

bgp-direct routes

bgp-direct redistribution supports policy via prefix lists and route-maps. This policy is applied to incoming original unicast routes before the redistribution translations (described below) are performed.

Redistribution of *bgp-direct* routes is performed in one of three possible modes: *plain*, *nve-group*, or *resolve-nve*. The default mode is *plain*. These modes indicate the kind of translations applied to routes before they are added to the VNC RIB.

In *plain* mode, the route's next hop is unchanged and the RD is set based on the next hop. For *bgp-direct* redistribution, the following translations are performed:

- The VN address is set to the original unicast route's next hop address.
- The UN address is NOT set. (VN->UN mapping will occur via ENCAP route or attribute, based on *vnc advertise-un-method* setting, generated by the RFP registration of the actual NVE)
- The RD is set to as if auto:vn:0 were specified (i.e., *rd_type=IP=1*:*IPv4-address=VN-address*:*two-byte-integer=0*)
- The RT list is included in the extended community list copied from the original unicast route (i.e., it must be set in the original unicast route).

In *nve-group* mode, routes are registered with VNC as if they came from an NVE in the nve-group designated in the *vnc redistribute nve-group* command. The following translations are performed:

- The next hop/VN address is set to the VN prefix configured for the redistribute nve-group.
- The UN address is set to the UN prefix configured for the redistribute nve-group.
- The RD is set to the RD configured for the redistribute nve-group.
- The RT list is set to the RT list configured for the redistribute nve-group. If *bgp-direct* routes are being redistributed, any extended communities present in the original unicast route will also be included.

In *resolve-nve* mode, the next hop of the original BGP route is typically the address of an NVE connected router (CE) connected by one or more NVEs. Each of the connected NVEs will register, via RFP, a VNC host route to the CE. This mode may be though of as a mechanism to proxy RFP registrations of BGP unicast routes on behalf of registering NVEs.

Multiple copies of the BGP route, one per matching NVE host route, will be added to VNC. In other words, for a given BGP unicast route, each instance of a RFP-registered host route to the unicast route's next hop will result in an instance of an imported VNC route. Each such imported VNC route will have a prefix equal to the original BGP unicast route's prefix, and a next hop equal to the next hop of the matching RFP-registered host route. If there is no RFP-registered host route to the next hop of the BGP unicast route, no corresponding VNC route will be imported.

The following translations are applied:

- The Next Hop is set to the next hop of the NVE route (i.e., the VN address of the NVE).
- The extended community list in the new route is set to the union of:
- Any extended communities in the original BGP route
  - Any extended communities in the NVE route

   - An added route-origin extended community with the next hop of the original BGP route is added to the new route. The value of the local administrator field defaults 5226 but may be configured by the user via the *roo-ec-local-admin* parameter.

- The Tunnel Encapsulation attribute is set to the value of the Tunnel Encapsulation attribute of the NVE route, if any.

bgp-direct-to-nve-groups routes

Unicast routes from the main or a designated instance of BGP may be redistributed to VNC as bgp-direct-to-nve-groups routes. These routes are NOT announced via BGP, but they are made available for local RFP lookup in response to queries from NVEs.

A non-main/default BGP instance is configured using the *router bgp AS view NAME* command as described elsewhere in this document.

In order for a route in the unicast BGP RIB to be made available to a querying NVE, there must already be, available to that NVE, an (interior) VNC route matching the next hop address of the unicast route. When the unicast route is provided to the NVE, its next hop is replaced by the next hop of the corresponding NVE. If there are multiple longest-prefix-match VNC routes, the unicast route will be replicated for each.

There is currently no policy (prefix-list or route-map) support for *bgp-direct-to-nve-groups* routes.

Redistribution Command Syntax

`vnc redistribute ipv4|ipv6 bgp|bgp-direct|ipv6 bgp-direct-to-nve-groups|connected|kernel|ospf|rip|stati`

`vnc redistribute ipv4|ipv6 bgp-direct-to-nve-groups view VIEWNAME`

Import (or do not import) prefixes from another routing protocols. Specify both the address family to import (*ipv4* or *ipv6*) and the protocol (*bgp*, *bgp-direct*, *bgp-direct-to-nve-groups*, *connected*, *kernel*, *ospf*, *rip*, or *static*). Repeat this statement as needed for each combination of address family and routing protocol. Prefixes from protocol *bgp-direct* are imported from unicast BGP in the same bgpd process. Prefixes from all other protocols (including *bgp*) are imported via the *zebra* kernel routing process.

`vnc redistribute mode plain|nve-group|resolve-nve`

Redistribute routes from other protocols into VNC using the specified mode. Not all combinations of modes and protocols are supported.

`vnc redistribute nve-group GROUP-NAME`

When using *nve-group* mode, assign (or do not assign) the NVE group *group-name* to routes redistributed from another routing protocol. *group-name* must be configured using *vnc nve-group*.

The VN and UN prefixes of the nve-group must both be configured, and each prefix must be specified as a full-length (/32 for IPv4, /128 for IPv6) prefix.

`vnc redistribute lifetime LIFETIME|infinite`

Assign a registration lifetime, either *lifetime* seconds or *infinite*, to prefixes redistributed from other routing protocols as if they had been received via RFP registration messages from an NVE. *lifetime* can be any integer between 1 and 4294967295, inclusive.

`vnc redistribute resolve-nve roo-ec-local-admin 0-65536`

Assign a value to the local-administrator subfield used in the Route Origin extended community that is assigned to routes exported under the *resolve-nve* mode. The default value is *5226*.

The following four *prefix-list* and *route-map* commands may be specified in the context of an nve-group or not. If they are specified in the context of an nve-group, they apply only if the redistribution mode is *nve-group*, and then only for routes being redistributed from *bgp-direct*. If they are specified outside the context of an nve-group, then they apply only for redistribution modes *plain* and *resolve-nve*, and then only for routes being redistributed from *bgp-direct*.

vnc redistribute bgp-direct (ipv4|ipv6) prefix-list LIST-NAME
When redistributing *bgp-direct* routes, specifies that the named prefix-list should be applied.

vnc redistribute bgp-direct no (ipv4|ipv6) prefix-list
When redistributing *bgp-direct* routes, specifies that no prefix-list should be applied.

vnc redistribute bgp-direct route-map MAP-NAME
When redistributing *bgp-direct* routes, specifies that the named route-map should be applied.

vnc redistribute bgp-direct no route-map
When redistributing *bgp-direct* routes, specifies that no route-map should be applied.

Configuring Export of Routes to Other Routing Protocols

Routes from VNC (both for RFP and for redistribution via BGP) can be provided to other protocols, either via zebra or directly to BGP.

It is important to note that when exporting routes to other protocols, the downstream protocol must also be configured to import the routes. For example, when VNC routes are exported to unicast BGP, the BGP configuration must include a corresponding *redistribute vnc-direct* statement.

export bgp|zebra mode none|group-nve|registering-nve|ce
Specify how routes should be exported to bgp or zebra. If the mode is *none*, routes are not exported. If the mode is *group-nve*, routes are exported according to nve-group or vrf-policy group configuration (*VNC NVE Group Configuration*): if a group is configured to allow export, then each prefix visible to the group is exported with next hops set to the currently-registered NVEs. If the mode is *registering-nve*, then all VNC routes are exported with their original next hops. If the mode is *ce*, only VNC routes that have an NVE connected CE Router encoded in a Route Origin Extended Community are exported. This extended community must have an administrative value that matches the configured *roo-ec-local-admin* value. The next hop of the exported route is set to the encoded NVE connected CE Router.

The default for both bgp and zebra is mode *none*.

vnc export bgp|zebra group-nve group GROUP-NAME

vnc export bgp|zebra group-nve no group GROUP-NAME
When export mode is *group-nve*, export (or do not export) prefixes from the specified nve-group or vrf-policy group to unicast BGP or to zebra. Repeat this statement as needed for each nve-group to be exported. Each VNC prefix that is exported will result in N exported routes to the prefix, each with a next hop corresponding to one of the N NVEs currently associated with the nve-group.

Some commands have a special meaning under certain export modes.

export bgp|zebra ipv4|ipv6 prefix-list LIST-NAME When export mode is *ce* or *registering-nve*, specifies that the named prefix-list should be applied to routes being exported to bgp or zebra. Prefix-lists for ipv4 and ipv6 are independent of each other.

export bgp|zebra no ipv4|ipv6 prefix-list When export mode is *ce* or *registering-nve*, specifies that no prefix-list should be applied to routes being exported to bgp or zebra.

export bgp|zebra route-map MAP-NAME When export mode is *ce* or *registering-nve*, specifies that the named route-map should be applied to routes being exported to bgp or zebra.

export bgp|zebra no route-map When export mode is *ce* or *registering-nve*, specifies that no route-map should be applied to routes being exported to bgp or zebra.

When the export mode is *group-nve*, policy for exported routes is specified per-NVE-group or vrf-policy group inside a *nve-group RFG-NAME* block via the following commands(*VNC NVE Group Configuration*):

export bgp|zebra route-map MAP-NAME This command is valid inside a *nve-group RFG-NAME* block. It specifies that the named route-map should be applied to routes being exported to bgp or zebra.

export bgp|zebra no route-map This command is valid inside a *nve-group RFG-NAME* block. It specifies that no route-map should be applied to routes being exported to bgp or zebra.

export bgp|zebra ipv4|ipv6 prefix-list LIST-NAME This command is valid inside a *nve-group RFG-NAME* block. It specifies that the named prefix-list filter should be applied to routes being exported to bgp or zebra. Prefix-lists for ipv4 and ipv6 are independent of each other.

export bgp|zebra no ipv4|ipv6 prefix-list This command is valid inside a *nve-group RFG-NAME* block. It specifies that no prefix-list filter should be applied to routes being exported to bgp or zebra.

3.21.2 Manual Address Control

The commands in this section can be used to augment normal dynamic VNC. The *add vnc* commands can be used to manually add IP prefix or Ethernet MAC address forwarding information. The *clear vnc* commands can be used to remove manually and dynamically added information.

add vnc prefix (A.B.C.D/M|X:X::X:X/M) vn (A.B.C.D|X:X::X:X) un (A.B.C.D|X:X::X:X) [cost (0-255)] [lifetime (infinite|(1-4294967295))] [local-next-hop (A.B.C.D|X:X::X:X) [local-cost (0-255)]]

Register an IP prefix on behalf of the NVE identified by the VN and UN addresses. The *cost* parameter provides the administrative preference of the forwarding information for remote advertisement. If omitted, it defaults to 255 (lowest preference). The *lifetime* parameter identifies the period, in seconds, that the information remains valid. If omitted, it defaults to *infinite*. The optional *local-next-hop* parameter is used to configure a nexthop to be used by an NVE to reach the prefix via a locally connected CE router. This information remains local to the NVA, i.e., not passed to other NVAs, and is only passed to registered NVEs. When specified, it is also possible to provide a *local-cost* parameter to provide a forwarding preference. If omitted, it defaults to 255 (lowest preference).

add vnc mac xx:xx:xx:xx:xx:xx virtual-network-identifier (1-4294967295) vn (A.B.C.D|X:X::X:X) un (A.B.C.D|X:X::X:X) [prefix (A.B.C.D/M|X:X::X:X/M)] [cost (0-255)] [lifetime (infinite|(1-4294967295))]

Register a MAC address for a logical Ethernet (L2VPN) on behalf of the NVE identified by the VN and UN addresses. The optional *prefix* parameter is to support enable IP address mediation for the given prefix. The *cost* parameter provides the administrative preference of the forwarding information. If omitted, it defaults to 255. The *lifetime* parameter identifies the period, in seconds, that the information remains valid. If omitted, it defaults to *infinite*.

clear vnc prefix (*|A.B.C.D/M|X:X::X:X/M) (*|[(vn|un) (A.B.C.D|X:X::X:X|*) [(un|vn) (A.B.C.D|X:X::X:X|*)] [mac xx:xx:xx:xx:xx:xx] [local-next-hop (A.B.C.D|X:X::X:X)])

Delete the information identified by prefix, VN address, and UN address. Any or all of these parameters may be wildcarded to (potentially) match more than one registration. The optional *mac* parameter specifies a layer-2 MAC address that must match the registration(s) to be deleted. The optional *local-next-hop* parameter is used to delete specific local nexthop information.

clear vnc mac (*|xx:xx:xx:xx:xx:xx) virtual-network-identifier (*|(1-4294967295)) (*|[(vn|un) (A.B.C.D|X:X::X:X|*) [(un|vn) (A.B.C.D|X:X::X:X|*)] [prefix (*|A.B.C.D/M|X:X::X:X/M)])

Delete mac forwarding information. Any or all of these parameters may be wildcarded to (potentially) match more than one registration. The default value for the *prefix* parameter is the wildcard value *.

clear vnc nve (*|((vn|un) (A.B.C.D|X:X::X:X) [(un|vn) (A.B.C.D|X:X::X:X)]))

Delete prefixes associated with the NVE specified by the given VN and UN addresses. It is permissible to specify only one of VN or UN, in which case any matching registration will be deleted. It is also permissible to specify * in lieu of any VN or UN address, in which case all registrations will match.

3.21.3 Other VNC-Related Commands

Note: VNC-Related configuration can be obtained via the *show running-configuration* command when in *enable* mode.

The following commands are used to clear and display Virtual Network Control related information:

`clear vnc counters`

Reset the counter values stored by the NVA. Counter values can be seen using the *show vnc* commands listed above. This command is only available in *enable* mode.

`show vnc summary`

Print counter values and other general information about the NVA. Counter values can be reset using the *clear vnc counters* command listed below.

`show vnc nves`

`show vnc nves vn|un ADDRESS`

Display the NVA's current clients. Specifying *address* limits the output to the NVEs whose addresses match *address*. The time since the NVA last communicated with the NVE, per-NVE summary counters and each NVE's addresses will be displayed.

`show vnc queries`

`show vnc queries PREFIX`

Display active Query information. Queries remain valid for the default Response Lifetime (*VNC Defaults Configuration*) or NVE-group Response Lifetime (*VNC NVE Group Configuration*). Specifying *prefix* limits the output to Query Targets that fall within *prefix*.

Query information is provided for each querying NVE, and includes the Query Target and the time remaining before the information is removed.

`show vnc registrations [all|local|remote|holddown|imported]`

`show vnc registrations [all|local|remote|holddown|imported] PREFIX`

Display local, remote, holddown, and/or imported registration information. Local registrations are routes received via RFP, which are present in the NVA Registrations Cache. Remote registrations are routes received via BGP (VPN SAFIs), which are present in the NVE-group import tables. Holddown registrations are local and remote routes that have been withdrawn but whose holddown timeouts have not yet elapsed. Imported information represents routes that are imported into NVA and are made available to querying NVEs. Depending on configuration, imported routes may also be advertised via BGP. Specifying *prefix* limits the output to the registered prefixes that fall within *prefix*.

Registration information includes the registered prefix, the registering NVE addresses, the registered administrative cost, the registration lifetime and the time since the information was registered or, in the case of Holddown registrations, the amount of time remaining before the information is removed.

`show vnc responses [active|removed]`

`show vnc responses [active|removed] PREFIX`

Display all, active and/or removed response information which are present in the NVA Responses Cache. Responses remain valid for the default Response Lifetime (*VNC Defaults Configuration*) or NVE-group Response Lifetime (*VNC NVE Group Configuration*.) When Removal Responses are enabled (*General VNC Configuration*), such responses are listed for the Response Lifetime. Specifying *prefix* limits the output to the addresses that fall within *prefix*.

Response information is provided for each querying NVE, and includes the response prefix, the prefix-associated registering NVE addresses, the administrative cost, the provided response lifetime and the time remaining before the information is to be removed or will become inactive.

show memory vnc
Print the number of memory items allocated by the NVA.

3.21.4 Example VNC and VNC-GW Configurations

Mesh NVA Configuration

This example includes three NVAs, nine NVEs, and two NVE groups. Note that while not shown, a single physical device may support multiple logical NVEs. *A three-way full mesh with three NVEs per NVA.* shows `code` `NVA-1` (192.168.1.100), `NVA 2` (192.168.1.101), and `NVA 3` (192.168.1.102), which are connected in a full mesh. Each is a member of the autonomous system 64512. Each NVA provides VNC services to three NVE clients in the 172.16.0.0/16 virtual-network address range. The 172.16.0.0/16 address range is partitioned into two NVE groups, `group1` (172.16.0.0/17) and `group2` (172.16.128.0/17).

Each NVE belongs to either NVE group `group1` or NVE group `group2`. The NVEs `NVE 1`, `NVE 2`, `NVE 4`, `NVE 7`, and `NVE 8` are members of the NVE group `group1`. The NVEs `NVE 3`, `NVE 5`, `NVE 6`, and `NVE 9` are members of the NVE group `group2`.

Each NVA advertises NVE underlay-network IP addresses using the Tunnel Encapsulation Attribute.

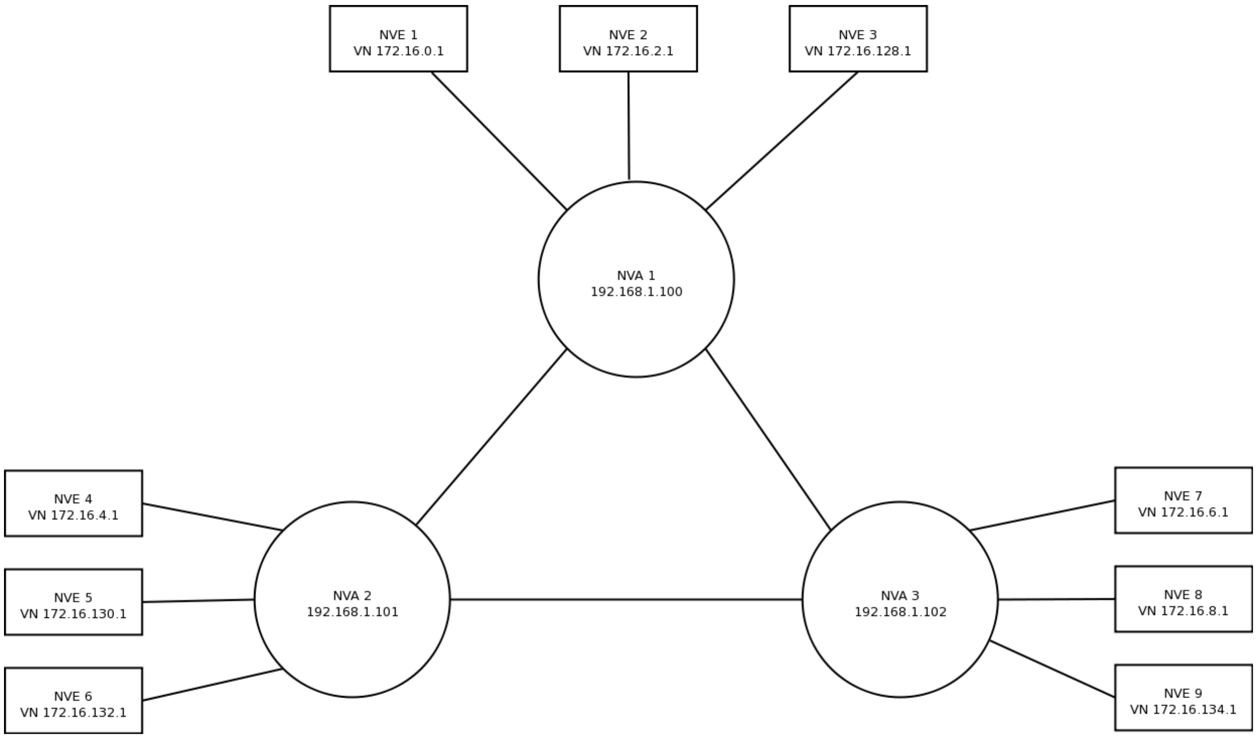


Fig. 6: A three-way full mesh with three NVEs per NVA.

`bgpd.conf` for `NVA 1` (192.168.1.100):

```
router bgp 64512
```

(continues on next page)

(continued from previous page)

```
    bgp router-id 192.168.1.100

    neighbor 192.168.1.101  remote-as 64512
    neighbor 192.168.1.102  remote-as 64512

    address-family ipv4 vpn
        neighbor 192.168.1.101 activate
        neighbor 192.168.1.102 activate
    exit-address-family

    vnc defaults
        rd 64512:1
        response-lifetime 200
        rt both 1000:1 1000:2
    exit-vnc

    vnc nve-group group1
        prefix vn 172.16.0.0/17
        rt both 1000:1
    exit-vnc

    vnc nve-group group2
        prefix vn 172.16.128.0/17
        rt both 1000:2
    exit-vnc

exit
```

bgpd.conf for NVA 2 (192.168.1.101):

```
router bgp 64512

    bgp router-id 192.168.1.101

    neighbor 192.168.1.100  remote-as 64512
    neighbor 192.168.1.102  remote-as 64512

    address-family ipv4 vpn
        neighbor 192.168.1.100 activate
        neighbor 192.168.1.102 activate
    exit-address-family

    vnc nve-group group1
        prefix vn 172.16.0.0/17
        rd 64512:1
        response-lifetime 200
        rt both 1000:1 1000:2
    exit-vnc
exit
```

bgpd.conf for NVA 3 (192.168.1.102):

```
router bgp 64512

    bgp router-id 192.168.1.102

    neighbor 192.168.1.101  remote-as 64512
    neighbor 192.168.1.102  remote-as 64512

    address-family ipv4 vpn
        neighbor 192.168.1.100 activate
        neighbor 192.168.1.101 activate
    exit-address-family

    vnc defaults
        rd 64512:1
        response-lifetime 200
        rt both 1000:1 1000:2
    exit-vnc

    vnc nve-group group1
        prefix vn 172.16.128.0/17
    exit-vnc
exit
```

Mesh NVA and VNC-GW Configuration

This example includes two NVAs, each with two associated NVEs, and two VNC-GWs, each supporting two CE routers physically attached to the four NVEs. Note that this example is showing a more complex configuration where VNC-GW is separated from normal NVA functions; it is equally possible to simplify the configuration and combine NVA and VNC-GW functions in a single FRR instance.

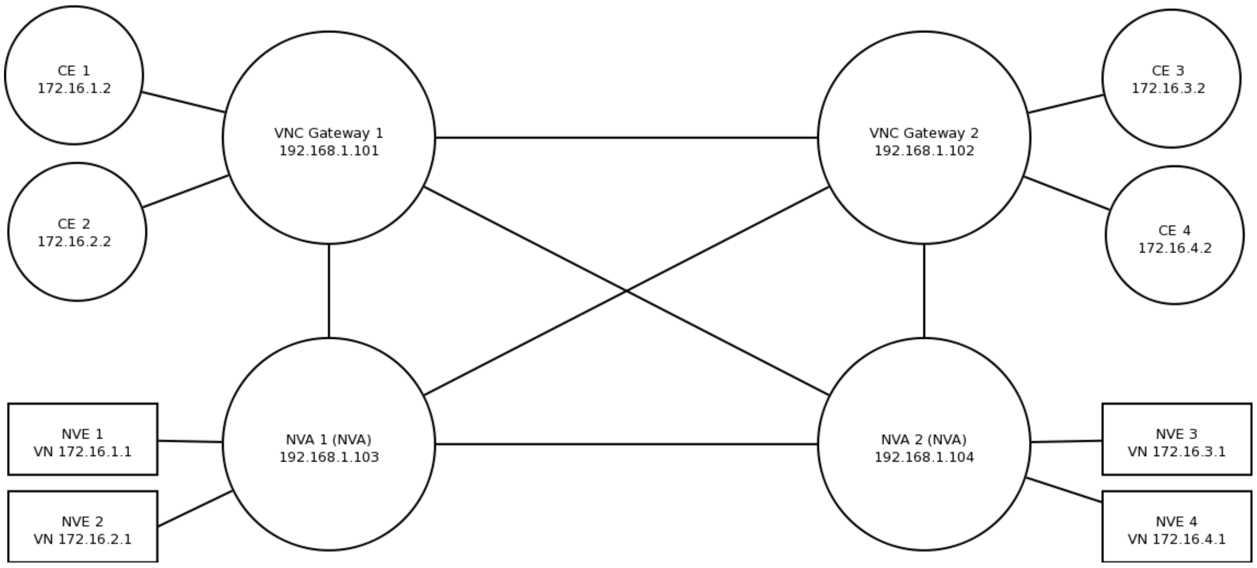


Fig. 7: Meshed NVEs and VNC-GWs

As shown in *Meshed NVEs and VNC-GWs*, NVAs and VNC-GWs are connected in a full iBGP mesh. The VNC-GWs each have two CEs configured as route-reflector clients. Each client provides BGP updates with unicast routes that the VNC-GW reflects to the other client. The VNC-GW also imports these unicast routes into VPN routes to be shared with the other VNC-GW and the two NVAs. This route importation is controlled with the `vnc redistribute` statements shown in the configuration. Similarly, registrations sent by NVEs via RFP to the NVAs are exported by the VNC-GWs to the route-reflector clients as unicast routes. RFP registrations exported this way have a next-hop address of the CE behind the connected (registering) NVE. Exporting VNC routes as IPv4 unicast is enabled with the `vnc export` command below.

The configuration for `VNC-GW 1` is shown below.

```
router bgp 64512
 bgp router-id 192.168.1.101
 bgp cluster-id 1.2.3.4
 neighbor 192.168.1.102 remote-as 64512
 neighbor 192.168.1.103 remote-as 64512
 neighbor 192.168.1.104 remote-as 64512
 neighbor 172.16.1.2 remote-as 64512
 neighbor 172.16.2.2 remote-as 64512
 !
 address-family ipv4 unicast
  redistribute vnc-direct
  no neighbor 192.168.1.102 activate
  no neighbor 192.168.1.103 activate
  no neighbor 192.168.1.104 activate
  neighbor 172.16.1.2 route-reflector-client
  neighbor 172.16.2.2 route-reflector-client
 exit-address-family
 !
 address-family ipv4 vpn
   neighbor 192.168.1.102 activate
   neighbor 192.168.1.103 activate
   neighbor 192.168.1.104 activate
 exit-address-family
 vnc export bgp mode ce
 vnc redistribute mode resolve-nve
 vnc redistribute ipv4 bgp-direct
 exit
```

Note that in the VNC-GW configuration, the neighboring VNC-GW and NVAs each have a statement disabling the IPv4 unicast address family. IPv4 unicast is on by default and this prevents the other VNC-GW and NVAs from learning unicast routes advertised by the route-reflector clients.

Configuration for `NVA 2`:

```
router bgp 64512
 bgp router-id 192.168.1.104
 neighbor 192.168.1.101 remote-as 64512
 neighbor 192.168.1.102 remote-as 64512
 neighbor 192.168.1.103 remote-as 64512
 !
 address-family ipv4 unicast
  no neighbor 192.168.1.101 activate
  no neighbor 192.168.1.102 activate
  no neighbor 192.168.1.103 activate
```

(continues on next page)

(continued from previous page)

```
exit-address-family
!
address-family ipv4 vpn
  neighbor 192.168.1.101 activate
  neighbor 192.168.1.102 activate
  neighbor 192.168.1.103 activate
exit-address-family
!
vnc defaults
 response-lifetime 3600
 exit-vnc
vnc nve-group nve1
 prefix vn 172.16.1.1/32
 response-lifetime 3600
 rt both 1000:1 1000:2
 exit-vnc
vnc nve-group nve2
 prefix vn 172.16.2.1/32
 response-lifetime 3600
 rt both 1000:1 1000:2
 exit-vnc
exit
```

VNC with FRR Route Reflector Configuration

A route reflector eliminates the need for a fully meshed NVA network by acting as the hub between NVAs. *Two NVAs and a BGP Route Reflector* shows BGP route reflector `BGP Route Reflector 1` (192.168.1.100) as a route reflector for NVAs `NVA 2``(192.168.1.101) and ``NVA 3` (192.168.1.102).

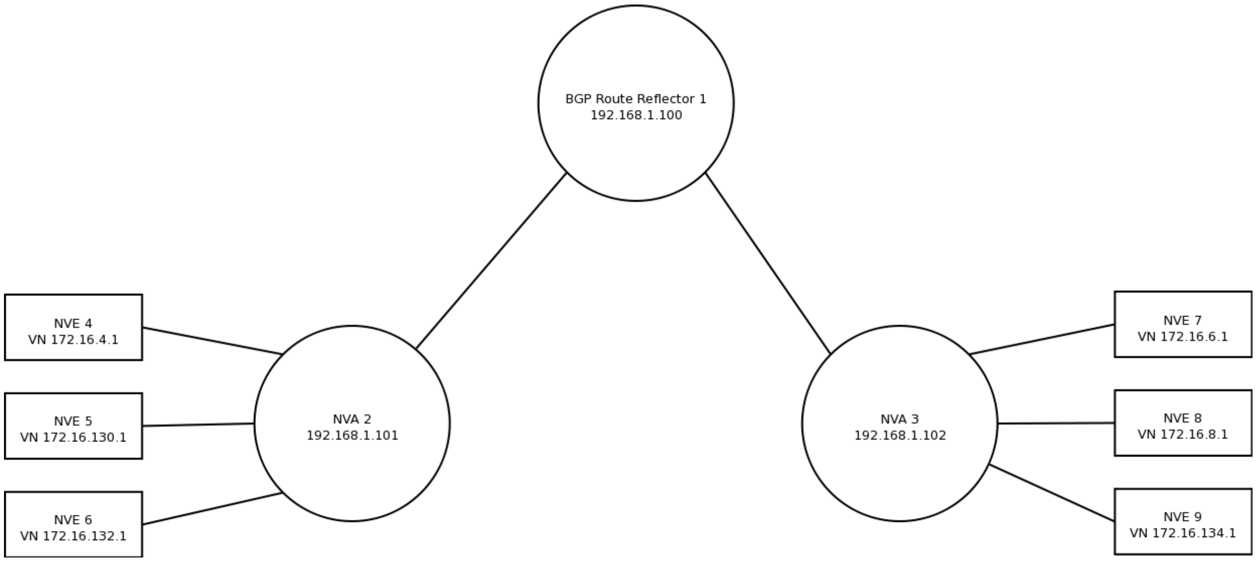


Fig. 8: Two NVAs and a BGP Route Reflector

`NVA 2` and `NVA 3` advertise NVE underlay-network IP addresses using the Tunnel Encapsulation Attribute. `BGP Route Reflector 1` reflects'' advertisements from ``NVA 2` to `NVA 3` and vice versa.

As in the example of *Mesh NVA Configuration*, there are two NVE groups. The 172.16.0.0/16 address range is partitioned into two NVE groups, `group1` (172.16.0.0/17) and `group2` (172.16.128.0/17). The NVE `NVE 4`, `NVE 7`, and `NVE 8` are members of the NVE group `group1`. The NVEs `NVE 5`, `NVE 6`, and `NVE 9` are members of the NVE group `group2`.

`bgpd.conf` for `BGP Route Reflector 1` on 192.168.1.100:

```
router bgp 64512

    bgp router-id 192.168.1.100

    neighbor 192.168.1.101 remote-as 64512
    neighbor 192.168.1.101 port 7179
    neighbor 192.168.1.101 description iBGP-client-192-168-1-101

    neighbor 192.168.1.102 remote-as 64512
    neighbor 192.168.1.102 port 7179
    neighbor 192.168.1.102 description iBGP-client-192-168-1-102

    address-family ipv4 unicast
        neighbor 192.168.1.101 route-reflector-client
        neighbor 192.168.1.102 route-reflector-client
    exit-address-family

    address-family ipv4 vpn
        neighbor 192.168.1.101 activate
        neighbor 192.168.1.102 activate

        neighbor 192.168.1.101 route-reflector-client
        neighbor 192.168.1.102 route-reflector-client
    exit-address-family

exit
```

`bgpd.conf` for `NVA 2` on 192.168.1.101:

```
router bgp 64512

    bgp router-id 192.168.1.101

    neighbor 192.168.1.100  remote-as 64512

    address-family ipv4 vpn
        neighbor 192.168.1.100 activate
    exit-address-family

    vnc nve-group group1
        prefix vn 172.16.0.0/17
        rd 64512:1
        response-lifetime 200
        rt both 1000:1 1000:2
    exit-vnc
exit
```

`bgpd.conf` for `NVA 2` on 192.168.1.102:

```
router bgp 64512

    bgp router-id 192.168.1.102

    neighbor 192.168.1.100  remote-as 64512

    address-family ipv4 vpn
        neighbor 192.168.1.100 activate
    exit-address-family

    vnc defaults
        rd 64512:1
        response-lifetime 200
        rt both 1000:1 1000:2
    exit-vnc

    vnc nve-group group1
        prefix vn 172.16.128.0/17
    exit-vnc
exit
```

While not shown, an NVA can also be configured as a route reflector.

VNC with Commercial Route Reflector Configuration

This example is identical to *VNC with FRR Route Reflector Configuration* with the exception that the route reflector is a commercial router. Only the VNC-relevant configuration is provided.

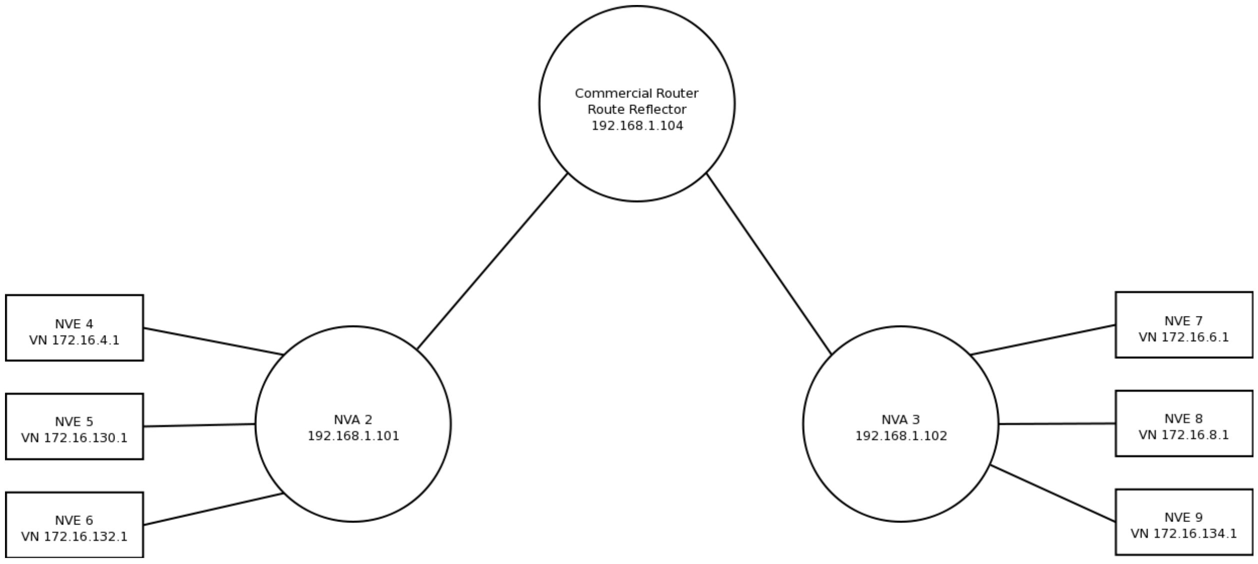


Fig. 9: Two NVAs with a commercial route reflector

`bgpd.conf` for BGP route reflector `Commercial Router` on 192.168.1.104::

```
version 8.5R1.13;
routing-options {
```

(continues on next page)

(continued from previous page)

```
    rib inet.0 {
        static {
            route 172.16.0.0/16 next-hop 192.168.1.104;
        }
    }
    autonomous-system 64512;
    resolution {
        rib inet.3 {
            resolution-ribs inet.0;
        }
        rib bgp.l3vpn.0 {
            resolution-ribs inet.0;
        }
    }
}
protocols {
    bgp {
        advertise-inactive;
        family inet {
            labeled-unicast;
        }
       group 1 {
            type internal;
            advertise-inactive;
            advertise-peer-as;
            import h;
            family inet {
                unicast;
            }
            family inet-vpn {
                unicast;
            }
            cluster 192.168.1.104;
            neighbor 192.168.1.101;
            neighbor 192.168.1.102;
        }
    }
}
policy-options {
    policy-statement h {
        from protocol bgp;
        then {
            as-path-prepend 64512;
            accept;
        }
    }
}
```

`bgpd.conf` for `NVA 2` on 192.168.1.101:

```
router bgp 64512
```

(continues on next page)

(continued from previous page)

```
    bgp router-id 192.168.1.101

    neighbor 192.168.1.100  remote-as 64512

    address-family ipv4 vpn
        neighbor 192.168.1.100 activate
    exit-address-family

    vnc nve-group group1
        prefix vn 172.16.0.0/17
        rd 64512:1
        response-lifetime 200
        rt both 1000:1 1000:2
    exit-vnc
exit
```

`bgpd.conf` for `NVA 3` on 192.168.1.102:

```
router bgp 64512

    bgp router-id 192.168.1.102

    neighbor 192.168.1.100  remote-as 64512

    address-family ipv4 vpn
        neighbor 192.168.1.100 activate
    exit-address-family

    vnc defaults
        rd 64512:1
        response-lifetime 200
        rt both 1000:1 1000:2
    exit-vnc

    vnc nve-group group1
        prefix vn 172.16.128.0/17
    exit-vnc
exit
```

VNC with Redundant Route Reflectors Configuration

This example combines the previous two (*VNC with FRR Route Reflector Configuration* and *VNC with Commercial Route Reflector Configuration*) into a redundant route reflector configuration. BGP route reflectors `BGP Route Reflector 1` and `Commercial Router` are the route reflectors for NVAs `NVA 2` and `NVA 3`. The two NVAs have connections to both route reflectors.

`bgpd.conf` for `BPGD Route Reflector 1` on 192.168.1.100:

```
router bgp 64512

 bgp router-id 192.168.1.100
 bgp cluster-id 192.168.1.100
```

(continues on next page)

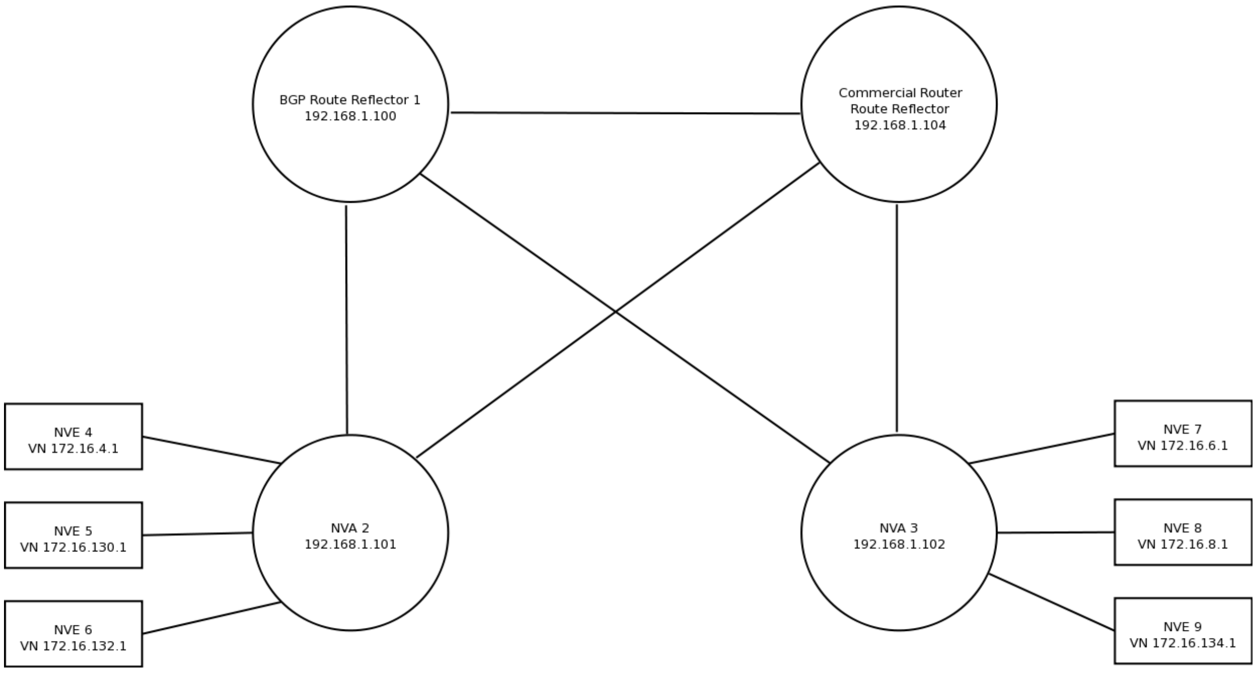


Fig. 10: FRR-based NVA with redundant route reflectors

(continued from previous page)

```
 neighbor 192.168.1.104 remote-as 64512

 neighbor 192.168.1.101 remote-as 64512
 neighbor 192.168.1.101 description iBGP-client-192-168-1-101
 neighbor 192.168.1.101 route-reflector-client

 neighbor 192.168.1.102 remote-as 64512
 neighbor 192.168.1.102 description iBGP-client-192-168-1-102
 neighbor 192.168.1.102 route-reflector-client

 address-family ipv4 vpn
  neighbor 192.168.1.101 activate
  neighbor 192.168.1.102 activate
  neighbor 192.168.1.104 activate

  neighbor 192.168.1.101 route-reflector-client
  neighbor 192.168.1.102 route-reflector-client
 exit-address-family
exit
```

`bgpd.conf` for `NVA 2` on 192.168.1.101:

```
router bgp 64512

 bgp router-id 192.168.1.101

 neighbor 192.168.1.100  remote-as 64512
```

(continues on next page)

(continued from previous page)

```
 neighbor 192.168.1.104  remote-as 64512

 address-family ipv4 vpn
  neighbor 192.168.1.100 activate
  neighbor 192.168.1.104 activate
 exit-address-family

 vnc nve-group group1
  prefix vn 172.16.0.0/17
  rd 64512:1
  response-lifetime 200
  rt both 1000:1 1000:2
 exit-vnc
exit
```

`bgpd.conf` for `NVA 3` on 192.168.1.102:

```
router bgp 64512

 bgp router-id 192.168.1.102

 neighbor 192.168.1.100  remote-as 64512
 neighbor 192.168.1.104  remote-as 64512

 address-family ipv4 vpn
  neighbor 192.168.1.100 activate
  neighbor 192.168.1.104 activate
 exit-address-family

 vnc defaults
  rd 64512:1
  response-lifetime 200
  rt both 1000:1 1000:2
 exit-vnc

 vnc nve-group group1
  prefix vn 172.16.128.0/17
 exit-vnc
exit
```

`bgpd.conf` for the Commercial Router route reflector on 192.168.1.104::

```
routing-options {
    rib inet.0 {
        static {
            route 172.16.0.0/16 next-hop 192.168.1.104;
        }
    }
    autonomous-system 64512;
    resolution {
        rib inet.3 {
            resolution-ribs inet.0;
```

(continues on next page)

(continued from previous page)

```
        }
        rib bgp.l3vpn.0 {
            resolution-ribs inet.0;
        }
    }
}
protocols {
    bgp {
        advertise-inactive;
        family inet {
            labeled-unicast;
        }
       group 1 {
            type internal;
            advertise-inactive;
            advertise-peer-as;
            import h;
            family inet {
                unicast;
            }
            family inet-vpn {
                unicast;
            }
            cluster 192.168.1.104;
            neighbor 192.168.1.101;
            neighbor 192.168.1.102;
        }

       group 2 {
            type internal;
            advertise-inactive;
            advertise-peer-as;
            import h;
            family inet {
                unicast;
            }
            family inet-vpn {
                unicast;
            }
            neighbor 192.168.1.100;
        }

    }
}
policy-options {
    policy-statement h {
        from protocol bgp;
        then {
            as-path-prepend 64512;
            accept;
        }
    }
```

(continues on next page)

(continued from previous page)

```
}
```

3.22 VRRP

VRRP stands for Virtual Router Redundancy Protocol. This protocol is used to allow multiple backup routers on the same segment to take over operation of each others' IP addresses if the primary router fails. This is typically used to provide fault-tolerant gateways to hosts on the segment.

FRR implements VRRPv2 (**RFC 3768**) and VRRPv3 (**RFC 5798**). For VRRPv2, no authentication methods are supported; these are deprecated in the VRRPv2 specification as they do not provide any additional security over the base protocol.

Note:

- VRRP is supported on Linux 5.1+
- VRRP does not implement Accept_Mode

3.22.1 Starting VRRP

The configuration file for *vrrpd* is `vrrpd.conf`. The typical location of `vrrpd.conf` is /etc/frr/vrrpd.conf.

If using integrated config, then `vrrpd.conf` need not be present and `frr.conf` is read instead.

VRRP supports all the common FRR daemon start options which are documented elsewhere.

3.22.2 Protocol Overview

From **RFC 5798**:

> VRRP specifies an election protocol that dynamically assigns responsibility for a virtual router to one of the VRRP routers on a LAN. The VRRP router controlling the IPv4 or IPv6 address(es) associated with a virtual router is called the Master, and it forwards packets sent to these IPv4 or IPv6 addresses. VRRP Master routers are configured with virtual IPv4 or IPv6 addresses, and VRRP Backup routers infer the address family of the virtual addresses being carried based on the transport protocol. Within a VRRP router, the virtual routers in each of the IPv4 and IPv6 address families are a domain unto themselves and do not overlap. The election process provides dynamic failover in the forwarding responsibility should the Master become unavailable. For IPv4, the advantage gained from using VRRP is a higher-availability default path without requiring configuration of dynamic routing or router discovery protocols on every end-host. For IPv6, the advantage gained from using VRRP for IPv6 is a quicker switchover to Backup routers than can be obtained with standard IPv6 Neighbor Discovery mechanisms.

VRRP accomplishes these goals primarily by using a virtual MAC address shared between the physical routers participating in a VRRP virtual router. This reduces churn in the neighbor tables of hosts and downstream switches and makes router failover theoretically transparent to these devices.

FRR implements the election protocol and handles changing the operating system interface configuration in response to protocol state changes.

As a consequence of the shared virtual MAC requirement, VRRP is currently supported only on Linux, as Linux is the only operating system that provides the necessary features in its network stack to make implementing this protocol feasible.

When a VRRP router is acting as the Master router, FRR allows the interface(s) with the backed-up IP addresses to remain up and functional. When the router transitions to Backup state, these interfaces are set into `protodown` mode. This is an interface mode that is functionally equivalent to `NO-CARRIER`. Physical drivers typically use this state indication to drop traffic on an interface. In the case of VRRP, the interfaces in question are macvlan devices, which are virtual interfaces. Since the IP addresses managed by VRRP are on these interfaces, this has the same effect as removing these addresses from the interface, but is implemented as a state flag.

3.22.3 Configuring VRRP

VRRP is configured on a per-interface basis, with some global defaults accessible outside the interface context.

System Configuration

FRR's VRRP implementation uses Linux macvlan devices to to implement the shared virtual MAC feature of the protocol. Currently, it does not create those system interfaces - they must be configured outside of FRR before VRRP can be enabled on them.

Each interface on which VRRP will be enabled must have at least one macvlan device configured with the virtual MAC and placed in the proper operation mode. The addresses backed up by VRRP are assigned to these interfaces.

Suppose you have an interface `eth0` with the following configuration:

```
$ ip addr show eth0
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group␣
↪default qlen 1000
    link/ether 02:17:45:00:aa:aa brd ff:ff:ff:ff:ff:ff
    inet 10.0.2.15/24 brd 10.0.2.255 scope global dynamic eth0
       valid_lft 72532sec preferred_lft 72532sec
    inet6 fe80::17:45ff:fe00:aaaa/64 scope link
       valid_lft forever preferred_lft forever
```

Suppose that the IPv4 and IPv6 addresses you want to back up are `10.0.2.16` and `2001:db8::370:7334`, and that they will be managed by the virtual router with id `5`. A macvlan device with the appropriate MAC address must be created before VRRP can begin to operate.

If you are using `ifupdown2`, the configuration is as follows:

```
iface eth0
 ...
 vrrp 5 10.0.2.16/24 2001:0db8::0370:7334/64
```

Applying this configuration with `ifreload -a` will create the appropriate macvlan device. If you are using `iproute2`, the equivalent configuration is:

```
ip link add vrrp4-2-1 link eth0 addrgenmode random type macvlan mode bridge
ip link set dev vrrp4-2-1 address 00:00:5e:00:01:05
ip addr add 10.0.2.16/24 dev vrrp4-2-1
ip link set dev vrrp4-2-1 up

ip link add vrrp6-2-1 link eth0 addrgenmode random type macvlan mode bridge
ip link set dev vrrp6-2-1 address 00:00:5e:00:02:05
ip addr add 2001:db8::370:7334/64 dev vrrp6-2-1
ip link set dev vrrp6-2-1 up
```

In either case, the created interfaces will look like this:

```
$ ip addr show vrrp4-2-1
5: vrrp4-2-1@eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP␣
↪group default qlen 1000
    link/ether 00:00:5e:00:01:05 brd ff:ff:ff:ff:ff:ff
    inet 10.0.2.16/24 scope global vrrp4-2-1
       valid_lft forever preferred_lft forever
    inet6 fe80::dc56:d11a:e69d:ea72/64 scope link stable-privacy
       valid_lft forever preferred_lft forever

$ ip addr show vrrp6-2-1
8: vrrp6-2-1@eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP␣
↪group default qlen 1000
 link/ether 00:00:5e:00:02:05 brd ff:ff:ff:ff:ff:ff
 inet6 2001:db8::370:7334/64 scope global
    valid_lft forever preferred_lft forever
 inet6 fe80::f8b7:c9dd:a1e8:9844/64 scope link stable-privacy
    valid_lft forever preferred_lft forever
```

Using `vrrp4-2-1` as an example, a few things to note about this interface:

- It is slaved to `eth0`; any packets transmitted on this interface will egress via `eth0`
- Its MAC address is set to the VRRP IPv4 virtual MAC specified by the RFC for VRID (Virtual Router ID) `5`
- The VIP (Virtual IP) address `10.0.2.16` must not be present on the parent interface `eth0`.
- The link local address on the interface is not derived from the interface MAC

First to note is that packets transmitted on this interface will egress via `eth0`, but with their Ethernet source MAC set to the VRRP virtual MAC. This is how FRR's VRRP implementation accomplishes the virtual MAC requirement on real hardware.

Ingress traffic is a more complicated matter. Macvlan devices have multiple operating modes that change how ingress traffic is handled. Of relevance to FRR's implementation are the `bridge` and `private` modes. In `private` mode, any ingress traffic on `eth0` (in our example) with a source MAC address equal to the MAC address on any of `eth0`'s macvlan devices will be placed *only* on that macvlan device. This curious behavior is undesirable, since FRR's implementation of VRRP needs to be able to receive advertisements from neighbors while in Backup mode - i.e., while its macvlan devices are in `protodown on`. If the macvlan devices are instead set to `bridge` mode, all ingress traffic shows up on all interfaces - including `eth0` - regardless of source MAC or any other factor. Consequently, macvlans used by FRR for VRRP must be set to `bridge` mode or the protocol will not function correctly.

As for the MAC address assigned to this interface, the last byte of the address holds the VRID, in this case `0x05`. The second to last byte is `0x01`, as specified by the RFC for IPv4 operation. The IPv6 MAC address is be identical except that the second to last byte is defined to be `0x02`. Two things to note from this arrangement:

1. There can only be up to 255 unique Virtual Routers on an interface (only 1 byte is available for the VRID)
2. IPv4 and IPv6 addresses must be assigned to different macvlan devices, because they have different MAC addresses

Finally, take note of the generated IPv6 link local address on the interface. For interfaces on which VRRP will operate in IPv6 mode, this link local *cannot* be derived using the usual EUI-64 method. This is because VRRP advertisements are sent from the link local address of this interface, and VRRP uses the source address of received advertisements as part of its election algorithm. If the IPv6 link local of a router is equivalent to the IPv6 link local in a received advertisement, this can cause both routers to assume the Master role (very bad). `ifupdown` knows to set the `addrgenmode` of the interface properly, but when using `iproute2` to create the macvlan devices, you must be careful to manually specify `addrgenmode random`.

A brief note on the Backup state

It is worth noting here that an alternate choice for the implementation of the Backup state, such as removing all the IP addresses assigned to the macvlan device or deleting their local routes instead of setting the device into `protodown on`, would allow the protocol to function regardless of whether the macvlan device(s) are set to `private` or `bridge` mode. Indeed, the strange behavior of the kernel macvlan driver in `private` mode, whereby it performs what may be thought of as a sort of interface-level layer 2 "NAT" based on source MAC, can be traced back to a patch clearly designed to accommodate a VRRP implementation from a different vendor. However, the `protodown` based implementation allows for a configuration model in which FRR does not dynamically manage the addresses assigned on a system, but instead just manages interface state. Such a scenario was in mind when this protocol implementation was initially built, which is why the other choices are not currently present. Since support for placing macvlan devices into `protodown` was not added to Linux until version 5.1, this also explains the relatively restrictive kernel versioning requirement.

In the future other methods of implementing Backup state may be added along with a configuration knob to choose between them.

Interface Configuration

Continuing with the example from the previous section, we assume the macvlan interfaces have been properly configured with the proper MAC addresses and the IPvX addresses assigned.

In FRR, a possible VRRPv3 configuration for this interface is:

```
interface eth0
 vrrp 5 version 3
 vrrp 5 priority 200
 vrrp 5 advertisement-interval 1500
 vrrp 5 ip 10.0.2.16
 vrrp 5 ipv6 2001:0db8::0370:7334
```

VRRP will activate as soon as the first IPvX address configuration line is encountered. If you do not want this behavior, use the *`vrrp (1-255) shutdown`* command, and apply the `no` form when you are ready to activate VRRP.

At this point executing `show vrrp` will display the following:

```
ubuntu-bionic# show vrrp

 Virtual Router ID                    5
 Protocol Version                     3
 Autoconfigured                       Yes
 Shutdown                             No
 Interface                            eth0
 VRRP interface (v4)                  vrrp4-2-5
 VRRP interface (v6)                  vrrp6-2-5
 Primary IP (v4)                      10.0.2.15
 Primary IP (v6)                      fe80::9b91:7155:bf6a:d386
 Virtual MAC (v4)                     00:00:5e:00:01:05
 Virtual MAC (v6)                     00:00:5e:00:02:05
 Status (v4)                          Master
 Status (v6)                          Master
 Priority                             200
 Effective Priority (v4)              200
 Effective Priority (v6)              200
 Preempt Mode                         Yes
```

(continues on next page)

(continued from previous page)

```
Accept Mode                          Yes
Advertisement Interval               1500 ms
Master Advertisement Interval (v4)   1000 ms
Master Advertisement Interval (v6)   1000 ms
Advertisements Tx (v4)               14
Advertisements Tx (v6)               14
Advertisements Rx (v4)               0
Advertisements Rx (v6)               0
Gratuitous ARP Tx (v4)               1
Neigh. Adverts Tx (v6)               1
State transitions (v4)               2
State transitions (v6)               2
Skew Time (v4)                       210 ms
Skew Time (v6)                       210 ms
Master Down Interval (v4)            3210 ms
Master Down Interval (v6)            3210 ms
IPv4 Addresses                       1
.................................    10.0.2.16
IPv6 Addresses                       1
.................................    2001:db8::370:7334
```

At this point, VRRP has sent gratuitous ARP requests for the IPv4 address, Unsolicited Neighbor Advertisements for the IPv6 address, and has asked Zebra to send Router Advertisements on its behalf. It is also transmitting VRRPv3 advertisements on the macvlan interfaces.

The Primary IP fields are of some interest, as the behavior may be counterintuitive. These fields show the source address used for VRRP advertisements. Although VRRPv3 advertisements are always transmitted on the macvlan interfaces, in the IPv4 case the source address is set to the primary IPv4 address on the base interface, `eth0` in this case. This is a protocol requirement, and IPv4 VRRP will not function unless the base interface has an IPv4 address assigned. In the IPv6 case the link local of the macvlan interface is used.

If any misconfiguration errors are detected, VRRP for the misconfigured address family will not come up and the configuration issue will be logged to FRR's configured logging destination.

Per the RFC, IPv4 and IPv6 virtual routers are independent of each other. For instance, it is possible for the IPv4 router to be in Backup state while the IPv6 router is in Master state; or for either to be completely inoperative while the other is operative, etc. Instances sharing the same base interface and VRID are shown together in the show output for conceptual convenience.

To complete your VRRP deployment, configure other routers on the segment with the exact same system and FRR configuration as shown above. Provided each router receives the others' VRRP advertisements, the Master election protocol will run, one Master will be elected, and the other routers will place their macvlan interfaces into `protodown on` until Master fails or priority values are changed to favor another router.

Switching the protocol version to VRRPv2 is accomplished simply by changing `version 3` to `version 2` in the VRID configuration line. Note that VRRPv2 does not support IPv6, so any IPv6 configuration will be rejected by FRR when using VRRPv2.

Note: All VRRP routers initially start in Backup state, and wait for the calculated Master Down Interval to pass before they assume Master status. This prevents downstream neighbor table churn if another router is already Master with higher priority, meaning this box will ultimately assume Backup status once the first advertisement is received. However, if the calculated Master Down Interval is high and this router is configured such that it will ultimately assume Master status, then it will take a while for this to happen. This is a known issue.

All interface configuration commands are documented below.

`vrrp (1-255) [version (2-3)]`
: Create a VRRP router with the specified VRID on the interface. Optionally specify the protocol version. If the protocol version is not specified, the default is VRRPv3.

`vrrp (1-255) advertisement-interval (10-40950)`
: Set the advertisement interval. This is the interval at which VRRP advertisements will be sent. Values are given in milliseconds, but must be multiples of 10, as VRRP itself uses centiseconds.

`vrrp (1-255) ip A.B.C.D`
: Add an IPv4 address to the router. This address must already be configured on the appropriate macvlan device. Adding an IP address to the router will implicitly activate the router; see `[no] vrrp (1-255) shutdown` to override this behavior.

`vrrp (1-255) ipv6 X:X::X:X`
: Add an IPv6 address to the router. This address must already be configured on the appropriate macvlan device. Adding an IP address to the router will implicitly activate the router; see `[no] vrrp (1-255) shutdown` to override this behavior.

  This command will fail if the protocol version is set to VRRPv2, as VRRPv2 does not support IPv6.

`vrrp (1-255) preempt`
: Toggle preempt mode. When enabled, preemption allows Backup routers with higher priority to take over Master status from the existing Master. Enabled by default.

`vrrp (1-255) checksum-with-ipv4-pseudoheader`
: Specify whether VRRPv3 checksum should involve IPv4 pseudoheader. This command should not affect VRRPv2 and IPv6. Enabled by default.

`vrrp (1-255) priority (1-254)`
: Set the router priority. The router with the highest priority is elected as the Master. If all routers in the VRRP virtual router are configured with the same priority, the router with the highest primary IP address is elected as the Master. Priority value 255 is reserved for the acting Master router.

`vrrp (1-255) shutdown`
: Place the router into administrative shutdown. VRRP will not activate for this router until this command is removed with the `no` form.

Global Configuration

Show commands, global defaults and debugging configuration commands.

`show vrrp [interface INTERFACE] [(1-255)] [json]`
: Shows VRRP status for some or all configured VRRP routers. Specifying an interface will only show routers configured on that interface. Specifying a VRID will only show routers with that VRID. Specifying `json` will dump each router state in a JSON array.

`debug vrrp [{protocol|autoconfigure|packets|sockets|ndisc|arp|zebra}]`
: Toggle debugging logs for VRRP components. If no component is specified, debugging for all components are turned on/off.

  protocol Logs state changes, election protocol decisions, and interface status changes.

  autoconfigure Logs actions taken by the autoconfiguration procedures. See *Autoconfiguration*.

  packets Logs details of ingress and egress packets. Includes packet decodes and hex dumps.

  sockets Logs details of socket configuration and initialization.

  ndisc Logs actions taken by the Neighbor Discovery component of VRRP.

arp Logs actions taken by the ARP component of VRRP.

zebra Logs communications with Zebra.

vrrp default <advertisement-interval (1-4096)|preempt|priority (1-254)|checksum-with-ipv4-pseudoheader|

Configure defaults for new VRRP routers. These values will not affect already configured VRRP routers, but will be applied to newly configured ones.

Autoconfiguration

In light of the complicated configuration required on the base system before VRRP can be enabled, FRR has the ability to automatically configure VRRP sessions by inspecting the interfaces present on the system. Since it is quite unlikely that macvlan devices with VRRP virtual MACs will exist on systems not using VRRP, this can be a convenient shortcut to automatically generate FRR configuration.

After configuring the interfaces as described in *System Configuration*, and configuring any defaults you may want, execute the following command:

vrrp autoconfigure [version (2-3)]

Generates VRRP configuration based on the interface configuration on the base system. If the protocol version is not specified, the default is VRRPv3. Any existing interfaces that are configured properly for VRRP - i.e. have the correct MAC address, link local address (when required), IPv4 and IPv6 addresses - are used to create a VRRP router on their parent interfaces, with VRRP IPvX addresses taken from the addresses assigned to the macvlan devices. The generated configuration appears in the output of `show run`, which can then be modified as needed and written to the config file. The `version` parameter controls the protocol version; if using VRRPv2, keep in mind that IPv6 is not supported and will not be configured.

The following configuration is then generated for you:

```
interface eth0
 vrrp 5
 vrrp 5 ip 10.0.2.16
 vrrp 5 ipv6 2001:db8::370:7334
```

VRRP is automatically activated. Global defaults, if set, are applied.

You can then edit this configuration with **vtysh** as needed, and commit it by writing to the configuration file.

Troubleshooting

My virtual routers are not seeing each others' advertisements

Check:

- Is your kernel at least 5.1?
- Did you set the macvlan devices to `bridge` mode?
- If using IPv4 virtual addresses, does the parent of the macvlan devices have an IPv4 address?
- If using IPv6 virtual addresses, is `addrgenmode` correctly set to `random` and not the default `eui64`?
- Is a firewall (`iptables`) or policy (`ip rule`) dropping multicast traffic?
- Do you have unusual `sysctls` enabled that could affect the operation of multicast traffic?
- Are you running in ESXi? See below.

My master router is not forwarding traffic

There's several possible causes here. If you're sure your configuration is otherwise correct, the following sysctl likely needs to be turned on:

```
sysctl -w net.ipv4.conf.eth0.ignore_routes_with_linkdown=1
```

Without this setting, it's possible to create topologies in which virtual routers holding mastership status will not forward traffic.

Issue reference: https://github.com/FRRouting/frr/issues/7391

My router is running in ESXi and VRRP isn't working

By default, ESXi traffic security settings don't allow traffic to egress a VNIC that does not have the MAC address assigned to the VNIC. This breaks VRRP, since virtual MACs are the basis of the protocol.

On ESXi before 6.7, you need to enable Promiscuous Mode in the ESXi settings. This is a significant security issue in some deployments so make sure you understand what you're doing. On 6.7 and later, you can use the MAC Learning feature instead, explained here.

Issue reference: https://github.com/FRRouting/frr/issues/5386

My router cannot interoperate with branded routers / L3 switches

FRR includes a pseudoheader when calculating VRRPv3 checksums by default, regardless of whether it's IPv4 or IPv6.

Some vendors have different interpretations of VRRPv3 RFC 5798 #5.2.8. In such cases, their checksums are calculated with a pseudoheader only when it comes to IPv6.

You need to disable `checksum-with-ipv4-pseudoheader` so that FRR computes and accepts such checksums.

Issue reference: https://github.com/FRRouting/frr/issues/9951

3.23 BMP

BMP (BGP Monitoring Protocol, **RFC 7854**) is used to send monitoring data from BGP routers to network management entities.

3.23.1 Implementation characteristics

The *BMP* implementation in FRR has the following properties:

- only the **RFC 7854** features are currently implemented. This means protocol version 3 without any extensions. It is not possible to use an older draft protocol version of BMP.
- the following statistics codes are implemented:
  - 0: count of prefixes rejected
  - 2: count of duplicate prefix withdrawals
  - 3: count of **prefixes** with loop in cluster id
  - 4: count of **prefixes** with loop in AS-path

  - 5: count of **prefixes** with loop in originator
  - 11: count of updates subjected to **RFC 7607** "treat as withdrawal" handling due to errors
  - 65531: *experimental* count of prefixes rejected due to invalid next-hop

  Note that stat items 3, 4 and 5 are specified to count updates, but FRR implements them as prefix-based counters.

- **route mirroring** is fully implemented, however BGP OPEN messages are not currently included in route mirroring messages. Their contents can be extracted from the "peer up" notification for sessions that established successfully. OPEN messages for failed sessions cannot currently be mirrored.
- **route monitoring** is available for IPv4 and IPv6 AFIs, unicast and multicast SAFIs. Other SAFIs (VPN, Labeled-Unicast, Flowspec, etc.) are not currently supported.
- monitoring peers that have BGP **add-path** enabled on the session will result in somewhat unpredictable behaviour. Currently, the outcome is:
  - route mirroring functions as intended, messages are copied verbatim
  - the add-path ID is never included in route monitoring messages
  - if multiple paths were received from a peer, an unpredictable path is picked and sent on the BMP session. The selection will differ for pre-policy and post-policy monitoring sessions.
  - as long as any path is present, something will be advertised on BMP sessions. Only after the last path is gone a withdrawal will be sent on BMP sessions.
  - updates to additional paths will trigger BMP route monitoring messages. There is no guarantee on consistency regarding which path is sent in these messages.
- monitoring peers with **RFC 5549** extended next-hops has not been tested.

3.23.2 Starting BMP

BMP is implemented as a loadable module. This means that to use BMP, `bgpd` must be started with the `-M bmp` option. It is not possible to enable BMP if `bgpd` was started without this option.

3.23.3 Configuring BMP

All of FRR's BMP configuration options are located inside the *`router bgp ASN`* block. Configure BGP first before proceeding to BMP setup.

There is one option that applies to the BGP instance as a whole:

`bmp mirror buffer-limit(0-4294967294)`

  This sets the maximum amount of memory used for buffering BGP messages (updates, keepalives, …) for sending in BMP Route Mirroring.

  The buffer is for the entire BGP instance; if multiple BMP targets are configured they reference the same buffer and do not consume additional memory. Queue overhead is included in accounting this memory, so the actual space available for BGP messages is slightly less than the value configured here.

  If the buffer fills up, the oldest messages are removed from the buffer and any BMP sessions where the now-removed messages were still pending have their **entire** queue flushed and a "Mirroring Messages Lost" BMP message is sent.

  BMP Route Monitoring is not affected by this option.

All other configuration is managed per targets:

bmp targets NAME
Create/delete a targets group. As implied by the plural name, targets may cover multiple outbound active BMP sessions as well as inbound passive listeners.

If BMP sessions have the same configuration, putting them in the same `bmp targets` will reduce overhead.

BMP session configuration

Inside a `bmp targets` block, the following commands control session establishment:

bmp connect HOSTNAME port (1-65535) {min-retry MSEC|max-retry MSEC} [source-interface WORD]
Add/remove an active outbound BMP session. HOSTNAME is resolved via DNS, if multiple addresses are returned they are tried in nondeterministic order. Only one connection will be established even if multiple addresses are returned. `min-retry` and `max-retry` specify (in milliseconds) bounds for exponential backoff. `source-interface` is the local interface on which the connection has to bind.

Warning: `ip access-list` and `ipv6 access-list` are checked for outbound connections resulting from `bmp connect` statements.

bmp listener <X:X::X:X|A.B.C.D> port (1-65535)
Accept incoming BMP sessions on the specified address and port. You can use `0.0.0.0` and `::` to listen on all IPv4/IPv6 addresses.

ip access-list NAME

ipv6 access-list NAME
Restrict BMP sessions to the addresses allowed by the respective access lists. The access lists are checked for both passive and active BMP sessions. Changes do not affect currently established sessions.

BMP data feed configuration

The following commands configure what BMP messages are sent on sessions associated with a particular `bmp targets`:

bmp stats [interval (100-86400000)]
Send BMP Statistics (counter) messages at the specified interval (in milliseconds.)

bmp monitor AFI SAFI <pre-policy|post-policy>
Perform Route Monitoring for the specified AFI and SAFI. Only IPv4 and IPv6 are currently valid for AFI. SAFI valid values are currently unicast, multicast, evpn and vpn. Other AFI/SAFI combinations may be added in the future.

All BGP neighbors are included in Route Monitoring. Options to select a subset of BGP sessions may be added in the future.

bmp mirror
Perform Route Mirroring for all BGP neighbors. Since this provides a direct feed of BGP messages, there are no AFI/SAFI options to be configured.

All BGP neighbors are included in Route Mirroring. Options to select a subset of BGP sessions may be added in the future.

3.24 WATCHFRR

WATCHFRR is a daemon that handles failed daemon processes and intelligently restarts them as needed.

3.24.1 Starting WATCHFRR

WATCHFRR is started as per normal systemd startup and typically does not require end users management.

3.24.2 WATCHFRR commands

`show watchfrr`

Give status information about the state of the different daemons being watched by WATCHFRR

`watchfrr ignore DAEMON`

Tell WATCHFRR to ignore a particular DAEMON if it goes unresponsive. This is particularly useful when you are a developer and need to debug a working system, without watchfrr pulling the rug out from under you.

3.25 MGMTd (Management Daemon)

The FRR Management Daemon (from now on referred to as MGMTd) is a new centralized entity representing the FRR Management Plane which can take management requests from any kind of UI/Frontend entity (e.g. CLI, Netconf, Restconf, Grpc etc.) over a new unified and common Frontend interface and can help maintain configurational data or retrieve operational data from any number of FRR managed entities/components that have been integrated with the new FRR Centralised Management Framework.

For organizing the management data to be owned by the FRR Management plane, management data is stored in YANG in compliance with a pre-defined set of YANG based schema. Data shall also be stored/retrieved in YANG format only.

The MGMTd also acts as a separate computational entity for offloading much of the management related computational overload involved in maintaining of management data and processing of management requests, from individual component daemons (which can otherwise be a signficant burden on the individual components, affecting performance of its other functionalities).

Lastly, the MGMTd works in-tandem with one (or more) MGMT Frontend Clients and a bunch of MGMT Backend Clients to realize the entirety of the FRR Management plane. Some of the advanatages of this new framework are:

1. Consolidation and management of all Management data by a single entity.
2. Better control over configuration validation, commit and rollback.
3. Faster collection of configuration data (without needing to involve individual component daemons).
4. Offload computational burden of YANG data parsing and validations of new configuration data being provisoned away from individual component daemons
5. Improve performance of individual component daemons while loading huge configuration or retrieving huge operational dataset.

The new FRR Management Daemon consists of the following sub-components:

- MGMT Frontend Interface
- MGMT Backend Interface
- MGMT Transaction Engine

3.25.1 MGMT Frontend Interface

The MGMT Frontend Interface is a bunch of message-based APIs that lets any UI/Frontend client to interact with the MGMT daemon to requests a set of management operations on a specific datastore/database. Following is a list of databases/datastores supported by the MGMT Frontend Interface and MGMTd:

- Candidate Database:
- Consists of configuration data items only.
- Data can be edited anytime using SET_CONFIG API.
- Data can be retrieved anytime using GET_CONFIG/GET_DATA API.
- Running Database:
- Consists of configuration data items only.
- Data cannot be edited using SET_CONFIG API.
- Data can only be modified using COMMIT_CONFIG API after which un-committed data from Candidate database will be first validated and applied to individualBackend component(s). Only on successful validation and apply on all individual components will the new data be copied over to the Running database.
- Data can be retrieved anytime using GET_CONFIG/GET_DATA API.
- Operational Database:
- Consists of non-configurational data items.
- Data is not stored on MGMT daemon. Rather it will be need to be fetched in real-time from the corresponding Backend component (if present).
- Data can be retrieved anytime using GET_DATA API.

Frontend Clients connected to MGMTd via Frontend Interface can themselves have multiple connections from one (or more) of its own remote clients. The MGMT Frontend Interface supports reresenting each of the remote clients for a given Frontend client(e.g. Netconf clients on a single Netconf server) as individual Frontend Client Sessions. So a single connection from a single Frontend Client can create more than one Frontend Client sessions.

Following are some of the management operations supported:

- INIT_SESSION/CLOSE_SESSION: Create/Destroy a session. Rest of all the operations are supported only in the context of a specific session.
- LOCK_DB/UNLOCK_DB: Lock/Unlock Management datastores/databases.
- GET_CONFIG/GET_DATA: Retrieve configurational/operational data from a specific datastore/database.
- SET_CONFIG/DELETE_CONFIG: Add/Modify/Delete specific data in a specific datastore/database.
- COMMIT_CONFIG: Validate and/or apply the uncommited set of configurations from one configuration database to another.
- Currently committing configurations from Candidate to Running database is only allowed, and not vice versa.

The exact set of message-based APIs are represented as Google Protobuf messages and can be found in the following file distributed with FRR codebase.

```
lib/mgmt.proto
```

The MGMT daemon implements a MGMT Frontend Server that opens a UNIX socket-based IPC channel on the following path to listen for incoming connections from all possible Frontend clients:

```
/var/run/frr/mgmtd_fe.sock
```

Each connection received from a Frontend client is managed and tracked as a MGMT Frontend adapter by the MGMT Frontend Adapter sub-component implemented by MGMTd.

To facilitate faster development/integration of Frontend clients with MGMT Frontend Interface, a C-based library has been developed. The API specification of this library can be found at:

```
lib/mgmt_fe_client.h
```

Following is a list of message types supported on the MGMT Frontend Interface:

- SESSION_REQ<Client-Connection-Id, Destroy>
- SESSION_REPLY<Client-Connection-Id, Destroy, Session-Id>
- LOCK_DB_REQ <Session-Id, Database-Id>
- LOCK_DB_REPLY <Session-Id, Database-Id>
- UNLOCK_DB_REQ <Session-Id, Database-Id>
- UNLOCK_DB_REPLY <Session-Id, Database-Id>
- GET_CONFIG_REQ <Session-Id, Database-Id, Base-Yang-Xpath>
- GET_CONFIG_REPLY <Session-Id, Database-Id, Base-Yang-Xpath, Yang-Data-Set>
- SET_CONFIG_REQ <Session-Id, Database-Id, Base-Yang-Xpath, Delete, …>
- SET_CONFIG_REPLY <Session-Id, Database-id, Base-Yang-Xpath, …, Status>
- COMMIT_CONFIG_REQ <Session-Id, Source-Db-Id, Dest-Db-Id>
- COMMIT_CONFIG_REPLY <Session-Id, Source-Db-id, Dest-Db-Id, Status>
- GET_DATA_REQ <Session-Id, Database-Id, Base-Yang-Xpath>
- GET_DATA_REPLY <Session-Id, Database-id, Base-Yang-Xpath, Yang-Data-Set>
- REGISTER_NOTIFY_REQ <Session-Id, Database-Id, Base-Yang-Xpath>
- DATA_NOTIFY_REQ <Database-Id, Base-Yang-Xpath, Yang-Data-Set>

Please refer to the MGMT Frontend Client Developers Reference and Guide (coming soon) for more details.

3.25.2 MGMTD Backend Interface

The MGMT Backend Interface is a bunch of message-based APIs that can be used by individual component daemons like BGPd, Staticd, Zebra to connect with MGMTd and utilize the new FRR Management Framework to let any Frontend clients to retrieve any operational data or manipulate any configuration data owned by the individual daemon component.

Like the MGMT Frontend Interface, the MGMT Backend Interface is is also comprised of the following:

- MGMT Backend Server (running on MGMT daemon)
- MGMT Backend Adapter (running on MGMT daemon)
- MGMT Backend client (running on Backend component daemons)

The MGMT Backend Client and MGMT Backend Adapter sub-component communicates using a specific set of message-based APIs.

The exact set of message-based APIs are represented as Google Protobuf messages and can be found in the following file distributed with FRR codebase.

```
lib/mgmt.proto
```

The MGMT daemon implements a MGMT Backend Server that opens a UNIX socket-based IPC channel on the following path to listen for incoming connections from all possible Backend clients:

```
/var/run/frr/mgmtd_be.sock
```

Each connection received from a Backend client is managed and tracked as a MGMT Backend adapter by the MGMT Backend Adapter sub-component implemented by MGMTd.

To facilitate faster development/integration of Backend clients with MGMTd, a C-based library has been developed. The API specification of this library can be found at:

```
lib/mgmt_be_client.h
```

Following is a list of message types supported on the MGMT Backend Interface:

- SUBSCRIBE_REQ <Req-Id, Base-Yang-Xpath, Filter-Type>
- SUBSCRIBE_REPLY <Req-Id, Status>
- TXN_REQ <Txn-Id, Create>
- TXN_REPLY <Txn-Id, Status>
- CREATE_CFGDATA_REQ <Txn-Id, Req-Id, Batch-Id, ConfigDataContents>
- CREATE_CFGDATA_ERROR <Txn-Id, Req-Id, Batch-Id, Status>
- VALIDATE_CFGDATA_REQ <Txn-Id, Batch-Id>
- VALIDATE_CFGDATA_REPLY <Txn-Id, Batch-Id, Status, ErrorInfo>
- APPLY_CFGDATA_REQ <Txn-Id, Batch-Id>
- APPLY_CFGDATA_REPLY <Txn-Id, Batch-Id, Status, ErrorInfo>
- GET_OPERDATA_REQ <Txn-Id, Base-Yang-Xpath, Filter-Type>
- GET_OPERDATA_REPLY <Txn-Id, OperDataContents>

Please refer to the MGMT Backend Client Developers Reference and Guide (coming soon) for more details.

3.25.3 MGMTD Transaction Engine

The MGMT Transaction sub-component is the main brain of the MGMT daemon that takes management requests from one (or more) Frontend Client translates them into transactions and drives them to completion in co-oridination with one (or more) Backend client daemons involved in the request.

A transaction can be seen as a set of management procedures executed over the Backend Interface with one (or more) individual Backend component daemons, as a result of some management request initiated from a specific Frontend client session. These group of operations on the Backend Interface with one (or more) individual components involved should be executed without taking any further management requests from other Frontend client sessions. To maintain this kind of atomic behavior a lock needs to be acquired (sometimes implicitly if not explicitly) by the corresponding Frontend client session, on the various datastores/databases involved in the management request being executed. The same datastores/databases need to be unlocked when all the procedures have been executed and the transaction is being closed.

Following are some of the transaction types supported by MGMT:

- Configuration Transactions
- Used to execute management operations like SET_CONFIG and COMMIT_CONFIG that involve writing/over-writing the contents of Candidate and Running databases.
- One (and only) can be created and be in-progress at any given time.
- Once initiated by a specific Frontend Client session and is still in-progress, all subsequent SET_CONFIG and COMMIT_CONFIG operations from other Frontend Client sessions will be rejected and responded with failure.
- Requires acquiring write-lock on Candidate (and later Running) databases.
- Show Transactions
- Used to execute management operations like GET_CONFIG and GET_DATA that involve only reading the contents of Candidate and Running databases (and sometimes real-time retrieval of operational data from individual component daemons).
- Multiple instance of this transaction type can be created and be in-progress at any given time.
- However, when a configuration transaction is currently in-progress show transaction can be initiated by any Frontend Client session.
- Requires acquiring read-lock on Candidate and/or Running databases.
- NOTE: Currently GET_DATA on Operational database is NOT supported. To be added in a future time soon.

3.25.4 MGMTD Configuration Rollback and Commit History

The MGMT daemon maintains upto 10 last configuration commit buffers and can rollback the contents of the Running Database to any of the commit-ids maintained in the commit buffers.

Once the number of commit buffers exceeds 10, the oldest commit buffer is deleted to make space for the latest commit. Also on rollback to a specific commit-id, buffer of all the later commits are deleted from commit record.

Configuration rollback is only allowed via VTYSH shell as of today and is not possible through the MGMT Frontend interface.

3.25.5 MGMT Configuration commands

`mgmt set-config XPATH VALUE`

This command uses a SET_CONFIG request over the MGMT Frontend Interface for the specified xpath with specific value. This command is used for testing purpose only. But can be used to set configuration data from CLI using SET_CONFIG operations.

`mgmt delete-config XPATH`

This command uses a SET_CONFIG request (with delete option) over the MGMT Frontend Interface o delete the YANG data node at the given xpath unless it is a key-leaf node(in which case it is not deleted).

`mgmt load-config FILE <merge|replace>`

This command loads configuration in JSON format from the filepath specified, and merges or replaces the Candidate DB as per the option specified.

`mgmt save-config <candidate|running> FILE`

This command dumps the DB specified in the db-name into the file in JSON format. This command in not supported for the Operational DB.

`mgmt commit abort`
This command will abort any configuration present on the Candidate but not been applied to the Running DB.

`mgmt commit apply`
This command commits any uncommited changes in the Candidate DB to the Running DB.

`mgmt commit check`
This command validates the configuration but does not apply them to the Running DB.

`mgmt rollback commit-id WORD`
This command rolls back the Running Database contents to the state corresponding to the commit-id specified.

`mgmt rollback last WORD`
This command rolls back the last specified number of recent commits.

3.25.6 MGMT Show commands

`show mgmt backend-adapter all`
This command shows the backend adapter information and the clients/daemons connected to the adapters.

`show mgmt backend-yang-xpath-registry`
This command shows which Backend adapters are registered for which YANG data subtree(s).

`show mgmt frontend-adapter all [detail]`
This command shows the frontend adapter information and the clients connected to the adapters.

`show mgmt transaction all`
Shows the list of transaction and bunch of information about the transaction.

`show mgmt get-config [candidate|running] XPATH`
This command uses the GET_CONFIG operation over the MGMT Frontend interface and returns the xpaths and values of the nodes of the subtree pointed by the <xpath>.

`show mgmt get-data [candidate|operation|running] XPATH`
This command uses the GET_DATA operation over the MGMT Frontend interface and returns the xpaths and values of the nodes of the subtree pointed by the <xpath>. Currenlty supported values for 'candidate' and 'running' only ('operational' shall be supported in future soon).

`show mgmt database-contents [candidate|operation|running] [xpath WORD] [file WORD] json|xml`
This command dumps the subtree pointed by the xpath in JSON or XML format. If filepath is not present then the tree will be printed on the shell.

`show mgmt commit-history`
This command dumps details of upto last 10 commits handled by MGMTd.

3.25.7 MGMT Daemon debug commands

The following debug commands enable debugging within the management daemon:

`[no] debug mgmt backend`
Enable[/Disable] debugging messages related to backend operations within the management daemon.

`[no] debug mgmt datastore`
Enable[/Disable] debugging messages related to YANG datastore operations within the management daemon.

`[no] debug mgmt frontend`
Enable[/Disable] debugging messages related to frontend operations within the management daemon.

`[no] debug mgmt transaction`
Enable[/Disable] debugging messages related to transactions within the management daemon.

3.25.8 MGMT Client debug commands

The following debug commands enable debugging within the management front and backend clients:

`[no] debug mgmt client backend`
    Enable[/Disable] debugging messages related to backend operations inside the backend mgmtd clients.

`[no] debug mgmt client frontend`
    Enable[/Disable] debugging messages related to frontend operations inside the frontend mgmtd clients.

RIFT

Routing in Fat Tree (RIFT) is a routing protocol with no operational expenses, designed for packet routing in CLOS-based and Fat Tree network topologies. This protocol blends link-state and distance-vector methods, offering multiple advantages for IP fabrics, including simplified management and enhanced network resilience.

riftd is based on draft-ietf-rift-rift-19.

Starting and Stopping riftd

The default configuration file name of *riftd*'s is `riftd.conf`. When invocation *riftd* searches directory /etc/frr. If `riftd.conf` is not there next search current directory.

RIFT uses UDP ports 914 and 915 to send and receive LIE/TIE packets. So the user must have the capability to bind the port, generally this means that the user must have superuser privileges. RIFT requires interface information maintained by *zebra* daemon. So running *zebra* is mandatory to run *riftd*. Thus minimum sequence for running RIFT is like below:

```
# zebra -d
# riftd -d
```

Please note that *zebra* must be invoked before *riftd*.

To stop *riftd*. Please use: kill `cat /var/run/frr/riftd.pid`

RIFT Configuration

```
router rift
```

The `router rift` command is necessary to enable RIFT. To disable RIFT, use the `no router rift` command. RIFT must be enabled before carrying out any of the RIFT commands.

```
system-id (1-4294967295)
```

Set RIFT's system ID.

```
level <1-20|leaf|ew|tof>
```

Clos and Fat Tree networks are topologically partially ordered graphs and `level` denotes the set of nodes at the same height in such a network. `leaf` means that the node is at the last level of the network. `ew` describes an East-West link. `tof` is a node at the top of the network (maximum level number). If not specified, the level is auto-negotiated by RIFT with southbound and northbound neighbours.

```
lie address A.B.C.D
```

This command overrides the default IPv4 multicast address used to send LIE packets.

```
no lie address A.B.C.D
```

This command removes the IPv4 multicast address configured to send LIE packets (fallbacks to default one).

```
lie address X:X::X:X
```

This command overrides the default IPv6 multicast address used to send LIE packets.

```
no lie address X:X::X:X
```

This command removes the IPv6 multicast address configured to send LIE packets (fallbacks to default one).

```
interface IFNAME [active-key (0-255)]
```

This command enables RIFT on the specified interface. Optionally, an active key can be passed to validate incoming packets. By default, it accepts all the packets without checking any key.

By default, RIFT is not enabled on all interfaces, you have to specify the `interface` command in order to enable it.

```
no interface IFNAME
```

This command disables RIFT on the specified interface.

Below is very simple RIFT configuration for a leaf node that uses LIE multicast address `224.0.0.150` and enables RIFT on interface `eth1` .

```
router rift
```

```
  system-id 1234
  level leaf
  lie address 224.0.0.150
  interface eth1
```

How to Announce RIFT route

```
redistribute <babel|bgp|connected|eigrp|isis|kernel|openfabric|ospf|sharp|
              static|table> [metric (0-16)] [route-map WORD]
```

Redistribute routes from other sources into RIFT.

If you want to specify RIFT static prefixes:

```
prefix <A.B.C.D/M|X:X::X:X/M>
```

Specify either a IPv4 or IPv6 prefix.

Below is very simple RIFT configuration for a leaf node that enables RIFT on interface `eth3` and announces a IPv6 prefix `ecaf::/64`.

```
router rift
  system-id 58964
  level leaf
  interface eth3
  prefix ecaf::/64
```

RIFT route-map

Usage of *riftd*'s `route-map` support.

Optional argument `route-map MAP_NAME` can be added to each redistribute statement. You have to specify the `direction` (either `northbound` or `southbound`).

```
redistribute static [route-map MAP_NAME direction <northbound|southbound>]
redistribute connected [route-map MAP_NAME direction <northbound|southbound>]
.....
```

Route-map statement (Route Maps) is needed to use `route-map` functionality.

Show RIFT Information

To display RIFT routes.

```
show rift
```

Show RIFT routes.

```
show rift tie-db [direction <northbound|southbound>]
```

The command display the content of the Topology Information Element Database (TIE-DB). If the `direction` is provided, only shows the TIE-DB in that direction.

```
show rift disaggregation
```

The command displays all information related to automatic disaggregation (positive and negative).

Sample configuration

```
ip prefix-list DEF_ONLY_4 permit 0.0.0.0/0
ip prefix-list DEF_ONLY_4 deny any

route-map FILTER_DEFAULT_V4 deny 10
    match ip address DEF_ONLY_4

router rift
  system-id 1337
  level tof
  interface eth0
  interface eth1
  interface eth2
  redistribute connected
  redistribute static route-map FILTER_DEFAULT_V4 direction northbound
  prefix 2001::/64
  prefix 200.0.0.0/24
```