FRR User Manual

Release latest

FRR

Jan 14, 2020

Contents

CHAPTER 1

Introduction

1.1 Overview

FRR is a routing software package that provides TCP/IP based routing services with routing protocols support such as BGP, RIP, OSPF, IS-IS and more (see *Supported Protocols vs. Platform*). FRR also supports special BGP Route Reflector and Route Server behavior. In addition to traditional IPv4 routing protocols, FRR also supports IPv6 routing protocols. With an SNMP daemon that supports the AgentX protocol, FRR provides routing protocol MIB read-only access (*SNMP Support*).

FRR uses an advanced software architecture to provide you with a high quality, multi server routing engine. FRR has an interactive user interface for each routing protocol and supports common client commands. Due to this design, you can add new protocol daemons to FRR easily. You can use FRR library as your program's client user interface.

FRR is distributed under the GNU General Public License.

FRR is a fork of Quagga.

1.1.1 About FRR

Today, TCP/IP networks are covering all of the world. The Internet has been deployed in many countries, companies, and to the home. When you connect to the Internet your packet will pass many routers which have TCP/IP routing functionality.

A system with FRR installed acts as a dedicated router. With FRR, your machine exchanges routing information with other routers using routing protocols. FRR uses this information to update the kernel routing table so that the right data goes to the right place. You can dynamically change the configuration and you may view routing table information from the FRR terminal interface.

Adding to routing protocol support, FRR can setup interface's flags, interface's address, static routes and so on. If you have a small network, or a stub network, or xDSL connection, configuring the FRR routing software is very easy. The only thing you have to do is to set up the interfaces and put a few commands about static routes and/or default routes. If the network is rather large, or if the network structure changes frequently, you will want to take advantage of FRR's dynamic routing protocol support for protocols such as RIP, OSPF, IS-IS or BGP.

Traditionally, UNIX based router configuration is done by *ifconfig* and *route* commands. Status of routing table is displayed by *netstat* utility. Almost of these commands work only if the user has root privileges. FRR has a different system administration method. There are two user modes in FRR. One is normal mode, the other is enable mode. Normal mode user can only view system status, enable mode user can change system configuration. This UNIX account independent feature will be great help to the router administrator.

Currently, FRR supports common unicast routing protocols, that is BGP, OSPF, RIP and IS-IS. Upcoming for MPLS support, an implementation of LDP is currently being prepared for merging. Implementations of BFD and PIM-SSM (IPv4) also exist, but are not actively being worked on.

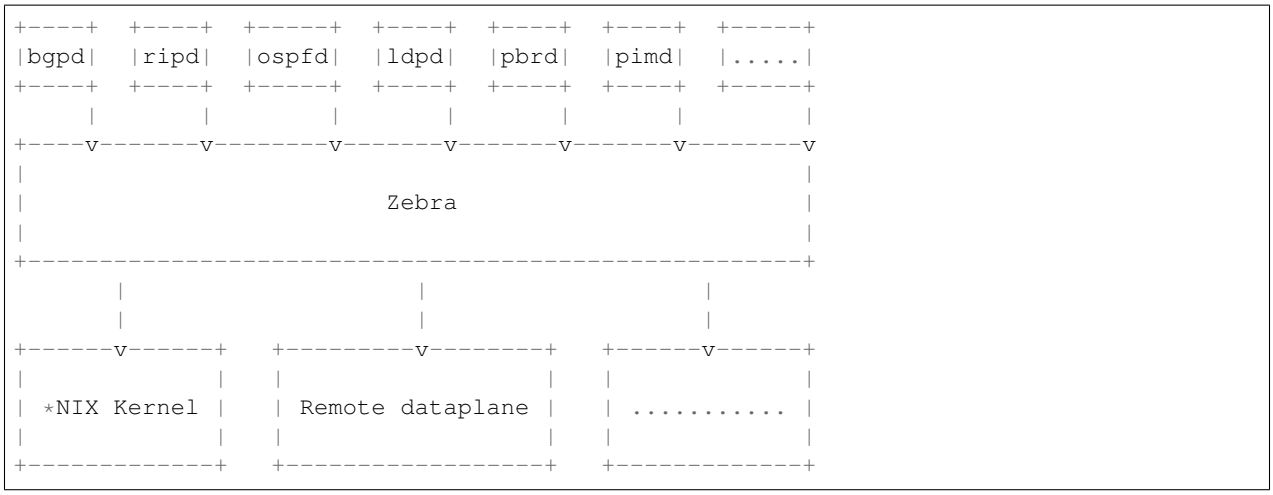
The ultimate goal of the FRR project is making a production-grade, high quality, featureful and free IP routing software suite.

1.1.2 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. There is a daemon for each major supported protocol as well as a middleman daemon (*Zebra*) which serves as the broker between these daemons and the kernel.

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.

An illustration of the large scale architecture is given below.



The multi-process architecture brings extensibility, modularity and maintainability. 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 avoiding the problem of having to maintain a separate configuration file for each daemon.

1.1.3 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:

- Solaris
- MacOS

Recent versions of the following compilers are well tested:

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

1.1.4 Supported Protocols vs. Platform

The following table lists all protocols cross-refrenced to all operating systems that have at least CI build tests. Note that for features, only features with system dependencies are included here.

Daemon / Feature	Linux	OpenBSD	FreeBSD	NetBSD	Solaris
FRR Core					
zebra	Y	Y	Y	Y	Y
VRF	4.8	N	N	N	N
MPLS	4.5	Y	N	N	N
pbrd (Policy Routing)	Y	N	N	N	N
WAN / Carrier protocols					
bgpd (BGP)	Y	Y	Y	Y	Y
VRF / L3VPN	4.8 †4.3	CP	CP	CP	CP
EVPN	4.18 †4.9	CP	CP	CP	CP
VNC (Virtual Network Control)	CP	CP	CP	CP	CP
Flowspec	CP	CP	CP	CP	CP
ldpd (LDP)	4.5	Y	N	N	N
VPWS / PW	N	5.8	N	N	N
VPLS	N	5.8	N	N	N
nhrpd (NHRP)	Y	N	N	N	N
Link-State Routing					
ospfd (OSPFv2)	Y	Y	Y	Y	Y
Segment Routing	4.12	N	N	N	N
ospf6d (OSPFv3)	Y	Y	Y	Y	Y
isisd (IS-IS)	Y	Y	Y	Y	Y
Distance-Vector Routing					
ripd (RIPv2)	Y	Y	Y	Y	Y
ripngd (RIPng)	Y	Y	Y	Y	Y
babeld (BABEL)	Y	Y	Y	Y	Y
eigrpd (EIGRP)	Y	Y	Y	Y	Y
Multicast Routing					
pimd (PIM)	4.18	N	Y	Y	Y

Continued on next page

Table 1 – continued from previous page

Daemon / Feature	Linux	OpenBSD	FreeBSD	NetBSD	Solaris
SSM (Source Specific)	Y	N	Y	Y	Y
ASM (Any Source)	Y	N	N	N	N
EVPN BUM Forwarding	5.0	N	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

Supported RFCs

FRR implements the following RFCs:

Note: This list is incomplete.

- **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.*
- **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 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 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 2858** *Multiprotocol Extensions for BGP-4. T. Bates, Y. Rekhter, R. Chandra, D. Katz. June 2000.*
- **RFC 2842** *Capabilities Advertisement with BGP-4. R. Chandra, J. Scudder. May 2000.*
- **RFC 3137** *OSPF Stub Router Advertisement, A. Retana, L. Nguyen, R. White, A. Zinin, D. McPherson. June 2001*
- **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 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.*
- **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 5883** *Bidirectional Forwarding Detection (BFD) for Multihop Paths, D. Katz, D. Ward. June 2010*

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.*

1.1.5 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.

1.1.6 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.

1.1.7 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.

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-systemd
Build watchfrr with systemd integration, this will allow FRR to communicate with systemd to tell systemd if FRR has come up properly.

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

--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.

--enable-datacenter
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 building of the example OSPF-API client.

--disable-ospf-ri
Disable support for OSPF Router Information (RFC4970 & RFC5088) this requires support for Opaque LSAs and Traffic Engineering.

--disable-isisd
Do not build isisd.

--disable-fabricd
Do not build fabricd.

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

--enable-isis-te
Enable Traffic Engineering Extension for ISIS (RFC5305)

--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.

--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.

--enable-fuzzing
Turn on some compile options to allow you to run fuzzing tools against the system. This flag is intended as a developer only tool and should not be used for normal operations.

--disable-snmp
Build without SNMP support.

--disable-vtysh
Build without VTYSH.

--enable-fpm
Build with FPM module support.

--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.

--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.

--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.

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.

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

--with-libyang-pluginsdir <dir>
Look for libyang plugins in *dir* [*prefix*/lib/frr/libyang_plugins]. Note that the FRR libyang plugins will be installed here.

This option is meaningless with libyang 0.16.74 or newer and will be removed once support for older libyang versions is dropped.

When it's desired to run FRR without installing it in the system, it's possible to configure it as follows to look for YANG modules and libyang plugins in the compile directory: .. code-block:: shell

./configure –with-libyang-pluginsdir="*pwd*/yang/libyang_plugins/.libs" –with-yangmodelsdir="*pwd*/yang"

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:

```
# 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 and improved upon through 4.13, which is the version most used in FRR testing (as of June 2018). Additional background on using Linux VRFs and kernel specific features can be found in http://schd.ws/hosted_files/ossna2017/fe/vrf-tutorial-oss.pdf.

The following impacts how BGP TCP sockets are managed across VRFs:

```
net.ipv4.tcp_l3mdev_accept=0
```

With this setting a BGP TCP socket is opened per VRF. This setting ensures that other TCP services, such as SSH, provided for non-VRF purposes are blocked from VRF associated Linux interfaces.

```
net.ipv4.tcp_l3mdev_accept=1
```

With this setting a single BGP TCP socket is shared across the system. This setting exposes any TCP service running on the system, e.g., SSH, to all VRFs. Generally this setting is not used in environments where VRFs are used to support multiple administrative groups.

Important note as of June 2018, Kernel versions 4.14-4.18 have a known bug where VRF-specific TCP sockets are not properly handled. When running these kernel versions, if unable to establish any VRF BGP adjacencies, either downgrade to 4.13 or set 'net.ipv4.tcp_l3mdev_accept=1'. The fix for this issue is planned to be included in future kernel versions. So upgrading your kernel may also address this issue.

Building

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

```
./configure \
    --prefix=/usr \
    --enable-exampledir=/usr/share/doc/frr/examples/ \
```

(continues on next page)

(continued from previous page)

```
    --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 Daemons 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
```

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

1.3.2 Daemons Configuration File

There is another file that controls the default options passed to daemons when starting FRR as a service. This file is located in your configuration directory, usually at `/etc/frr/daemons`.

This file has several parts. Here is an example:

```
#
# 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"
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"

# 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:

```
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.

```
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
```

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.

CHAPTER 2

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.

Each of the daemons has its own config file. The daemon name plus `.conf` is the default config file name. For example, zebra's default config file name is `zebra.conf`. You can specify a config file using the *-f* or *--config_file* options when starting the daemon.

Basic Config Commands

hostname HOSTNAME
Set hostname of the router.

[no] 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.

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

[no] 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.

[no] 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`.

[no] 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.

Note: If you do not configure any file logging, and a daemon crashes due to a signal or an assertion failure, it will attempt to save the crash information in a file named `/var/tmp/frr.<daemon name>.crashlog`. For security reasons, this will not happen if the file exists already, so it is important to delete the file after reporting the crash information.

[no] 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.

[no] log monitor [LEVEL]
Enable logging output to vty terminals that have enabled logging using the `terminal monitor` command. By default, monitor logging is enabled at the debugging level, but this command (or the deprecated `log trap` command) can be used to change the monitor logging level. 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 terminal monitors.

[no] 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.

[no] 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 (including Solaris) can be configured to include the facility and level in the messages emitted.

[no] 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. Once set, command logging can only be turned off by restarting the daemon.

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.

no banner motd
No motd banner string will be printed.

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.

no exec-timeout
Do not perform timeout at all. This command is as same as `exec-timeout 0 0`.

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

Sample Config File

Below is a sample configuration file for the zebra daemon.

```
!
! Zebra configuration file
!
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`.

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 memory
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
```

(continues on next page)

(continued from previous page)

```
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.

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

find COMMAND...
This command performs a simple substring 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
```

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":

```
frr# find l2vpn
  (view)  show [ip] bgp l2vpn evpn [json]
  (view)  show [ip] bgp l2vpn evpn all <A.B.C.D|A.B.C.D/M> [json]
  (view)  show [ip] bgp l2vpn evpn all neighbors A.B.C.D advertised-routes [json]
  (view)  show [ip] bgp l2vpn evpn all neighbors A.B.C.D routes [json]
  (view)  show [ip] bgp l2vpn evpn all overlay
  ...
```

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.

-v, --version
Print program version.

--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|debug
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.

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 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.

2.2.1 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.

[no] 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.

2.2.2 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.

2.2.3 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. Further, it is better not to use the integrated `frr.conf` file, as any syntax error in it can lead to /all/ of your daemons being unable to start up. Per daemon files are more robust as impact of errors in configuration are limited to the daemon in whose file the error is made.

service integrated-vtysh-config

no 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.

2.3 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.

2.3.1 IP Access List

access-list NAME permit IPV4-NETWORK

access-list NAME deny IPV4-NETWORK

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
```

2.3.2 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.

no ip prefix-list NAME

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.

no ip prefix-list NAME description [DESC]
Deletes the description from a prefix list. It is possible to use the command without the full description.

ip prefix-list sequential number control

ip prefix-list sequence-number
With this command, the IP prefix list sequential number is displayed. This is the default behavior.

no ip prefix-list sequence-number
With this command, the IP prefix list sequential number is not displayed.

Showing ip prefix-list

show ip prefix-list
Display all IP prefix lists.

show ip prefix-list NAME
Show IP prefix list can be used with a prefix list name.

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

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

show ip prefix-list summary NAME

show ip prefix-list detail

show ip prefix-list detail NAME

Clear counter of ip prefix-list

clear ip prefix-list
Clears the counters of all IP prefix lists. Clear IP Prefix List can be used with a specified name and prefix.

clear ip prefix-list NAME

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

2.4 Route Maps

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

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

2.4.1 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*.

2.4.2 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 IPV4_ADDR
Matches the specified *ipv4_addr*.

match aspath 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 source-protocol PROTOCOL_NAME
This is a ZEBRA 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.

2.4.3 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.

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 weight WEIGHT
Set the route's weight.

set metric METRIC
Set the BGP attribute MED.

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

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.

2.4.4 Route Map Call Command

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

2.4.5 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

2.4.6 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.

2.5 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.

2.5.1 Router Advertisement

no ipv6 nd suppress-ra
Send router advertisement messages.

ipv6 nd suppress-ra
Don't send router advertisement messages.

ipv6 nd prefix ipv6prefix [valid-lifetime] [preferred-lifetime] [off-link] [no-autoconfig]
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.

[no] ipv6 nd ra-interval [(1-1800)]
The maximum time allowed between sending unsolicited multicast router advertisements from the interface, in seconds. Default: `600`

[no] ipv6 nd ra-interval [msec (70-1800000)]
The maximum time allowed between sending unsolicited multicast router advertisements from the interface, in milliseconds. Default: `600000`

[no] 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`

[no] 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`

[no] 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

[no] 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

[no] 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

[no] 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`

[no] 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`

[no] 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

[no] ipv6 nd router-preference [(high|medium|low)]
Set default router preference in IPv6 router advertisements per RFC4191. Default: medium

[no] 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.

[no] 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

[no] 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

2.5.2 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)

2.6 Kernel Interface

There are several different methods for reading kernel routing table information, updating kernel routing tables, and for looking up interfaces.

- **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** On recent Linux kernels (2.0.x and 2.2.x), there is a kernel/user communication support called *Netlink*. It makes asynchronous communication between kernel and FRR possible, similar to a routing socket on BSD systems.

  Before you use this feature, be sure to select (in kernel configuration) the kernel/Netlink support option 'Kernel/User network link driver' and 'Routing messages'.

  Today, the `/dev/route` special device file is obsolete. Netlink communication is done by reading/writing over Netlink socket.

  After the kernel configuration, please reconfigure and rebuild FRR. You can use Netlink as a dynamic routing update channel between FRR and the kernel.

2.7 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.

2.7.1 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.

2.7.2 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
[...]
```

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

no agentx

2.7.3 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" ;;
5) peerstate="Openconfirm" ;;
6) peerstate="Established" ;;
```

(continues on next page)

(continued from previous page)

```
*) 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=""
;;
05)
error="Finite State Machine Error"
suberror=""
;;
06)
error="Cease"
```

(continues on next page)

(continued from previous page)

```
case "$suberrorcode" in
01) suberror="Maximum Number of Prefixes Reached" ;;
02) suberror="Administratively Shutdown" ;;
03) suberror="Peer Unconfigured" ;;
04) suberror="Administratively Reset" ;;
05) suberror="Connection Rejected" ;;
06) suberror="Other Configuration Change" ;;
07) suberror="Connection collision resolution" ;;
08) suberror="Out of Resource" ;;
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
echo "$MAIL" | mail -s "$SUBJECT" $EMAILADDR
```

CHAPTER 3

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, --keep_kernel
When zebra starts up, don't delete old self inserted routes.

-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

-o, --vrfdefaultname
When *Zebra* starts with this option, the default VRF name is changed to the parameter.

See also:

Virtual Routing and Forwarding

--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.

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

no shutdown
Up or down the current interface.

ip address ADDRESS/PREFIX

ipv6 address ADDRESS/PREFIX

no ip address ADDRESS/PREFIX

no ipv6 address ADDRESS/PREFIX
Set the IPv4 or IPv6 address/prefix for the interface.

ip address LOCAL-ADDR peer PEER-ADDR/PREFIX

no 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.

multicast

no multicast
Enable or disables multicast flag for the interface.

bandwidth (1-10000000)

no 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

no link-detect
Enable/disable link-detect on platforms which support this. Currently only Linux and Solaris, 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

link-params

no link-param
Enter into the link parameters sub node. At least 'enable' must be set to activate the link parameters, and consequently 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. Disable link parameters for this interface.

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

link-params [enable]
Enable link parameters for this interface.

link-params [metric (0-4294967295)]

link-params max-bw BANDWIDTH

link-params max-rsv-bw BANDWIDTH

link-params unrsv-bw (0-7) BANDWIDTH

link-params admin-grp 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 BANDIWDTH is specified in IEEE floating point format and express in Bytes/second.

link-param delay (0-16777215) [min (0-16777215) | max (0-16777215)]

link-param delay-variation (0-16777215)

link-param packet-loss PERCENTAGE

link-param res-bw BANDWIDTH

link-param ava-bw BANDWIDTH

link-param 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.

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

link-param no neighbor
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).

ip nht resolve-via-default
Allows nexthop tracking to resolve via the default route. This is useful when e.g. you want to allow BGP to peer across the default route.

3.1.4 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*.

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.

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.

By using the *-n* option, the *Linux network namespace* will be mapped over the *Zebra* VRF. One nice feature that is possible by handling *Linux network namespace* is the ability to name default VRF. At startup, *Zebra* discovers the available *Linux network namespace* by parsing folder */var/run/netns*. Each file stands for a *Linux network namespace*, but not all *Linux network namespaces* are available under that folder. This is the case for default VRF. It is possible to name the default VRF, by creating a file, by executing following commands.

```
touch /var/run/netns/vrf0
mount --bind /proc/self/ns/net /var/run/netns/vrf0
```

Above command illustrates what happens when the default VRF is visible under */var/run/netns/*. Here, the default VRF file is *vrf0*. At startup, FRR detects the presence of that file. It detects that the file statistics information matches the same file statistics information as */proc/self/ns/net* (through stat() function). As statistics information matches, then *vrf0* stands for the new default namespace name. Consequently, the VRF naming *Default* will be overridden by the new discovered namespace name *vrf0*.

For those who don't use VRF backend with *Linux network namespace*, it is possible to statically configure and recompile FRR. It is possible to choose an alternate name for default VRF. Then, the default VRF naming will automatically be updated with the new name. To illustrate, if you want to recompile with *global* value, use the following command:

```
./configure --with-defaultvrfname=global
```

3.1.5 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:

[no] 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.

[no] 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
```

3.1.6 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

no 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 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 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]

no 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.

3.1.7 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** or one of

- system,
- kernel,
- connected,
- static,
- rip,
- ripng,
- ospf,

- ospf6,
- isis,
- bgp,
- hsls.

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
```

3.1.8 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 FIB push interface comprises of a TCP connection between zebra and the FPM. The connection is initiated by zebra – that is, the FPM acts as the TCP server.

The relevant zebra code kicks in when zebra is configured with the `--enable-fpm` flag. Zebra periodically attempts to connect to the well-known FPM port. 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 encapsulation header for the messages exchanged with the FPM is defined by the file `fpm/fpm.h` in the frr tree. The routes themselves are encoded in Netlink or protobuf format, with Netlink being the default.

Protobuf is one of a number of new serialization formats wherein the message schema is expressed in a purpose-built language. Code for encoding/decoding to/from the wire format is generated from the schema. Protobuf messages can be extended easily while maintaining backward-compatibility with older code. Protobuf has the following advantages over Netlink:

- Code for serialization/deserialization is generated automatically. This reduces the likelihood of bugs, allows third-party programs to be integrated quickly, and makes it easy to add fields.
- The message format is not tied to an OS (Linux), and can be evolved independently.

As mentioned before, zebra encodes routes sent to the FPM in Netlink format by default. The format can be controlled via the FPM module's load-time option to zebra, which currently takes the values *Netlink* and *protobuf*.

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.

3.1.9 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.

3.1.10 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 interface

show ip prefix-list [NAME]

show route-map [NAME]

show ip protocol

show ipforward
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 ipv6forward
Display whether the host's IP v6 forwarding is enabled or not.

show zebra
Display various statistics related to the installation and deletion of routes, neighbor updates, and LSP's into the kernel.

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 choosen 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 zebra fpm stats
Display statistics related to the zebra code that interacts with the optional Forwarding Plane Manager (FPM) component.

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

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 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"

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 NA
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. This option conflicts with *vrf*.

vrf selects which domain we want to use.

no peer <A.B.C.D|X:X::X:X>$peer [{multihop|local-address <A.B.C.D|X:X::X:X>$local|interface
Stops and removes the selected peer.

show bfd peers [json]
Show all configured BFD peers information and current status.

show bfd peer <WORD$label|<A.B.C.D|X:X::X:X>$peer [{multihop|local-address <A.B.C.D|X:X::X
Show status for a specific BFD peer.

Peer Configurations

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.

echo-interval (10-60000)
Configures the minimal echo receive transmission interval that this system is capable of handling.

[no] echo-mode
Enables or disables the echo transmission mode. This mode is disabled by default.

It is recommended that the transmission interval of control packets to be increased after enabling echo-mode to reduce bandwidth usage. For example: *transmission-interval 2000*.

Echo mode is not supported on multi-hop setups (see **RFC 5883** section 3).

[no] shutdown
Enables or disables the peer. When the peer is disabled an 'administrative down' message is sent to the remote peer.

label WORD
Labels a peer with the provided word. This word can be referenced later on other daemons to refer to a specific peer.

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.

no neighbor <A.B.C.D|X:X::X:X|WORD> bfd
Removes any notification registration for this neighbor.

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.

no ip ospf bfd
Removes any notification registration for this interface peers.

OSPF6 BFD Configuration

The following commands are available inside the interface configuration node.

ipv6 ospf6 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.

no ipv6 ospf6 bfd
Removes any notification registration for this interface peers.

PIM BFD Configuration

The following commands are available inside the interface configuration node.

ip pim 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.

no ip pim bfd
Removes any notification registration for this interface peers.

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
 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 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
 !

 ! 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
```

(continues on next page)

(continued from previous page)

```
                ID: 1
                Remote ID: 1
                Status: up
                Uptime: 1 minute(s), 51 second(s)
                Diagnostics: ok
                Remote diagnostics: ok
                Local timers:
                        Receive interval: 300ms
                        Transmission interval: 300ms
                        Echo transmission interval: disabled
                Remote timers:
                        Receive interval: 300ms
                        Transmission interval: 300ms
                        Echo transmission 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
                Local timers:
                        Receive interval: 300ms
                        Transmission interval: 300ms
                        Echo transmission interval: disabled
                Remote timers:
                        Receive interval: 300ms
                        Transmission interval: 300ms
                        Echo transmission 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
                Local timers:
                        Receive interval: 300ms
                        Transmission interval: 300ms
                        Echo transmission interval: disabled
                Remote timers:
                        Receive interval: 300ms
                        Transmission interval: 300ms
                        Echo transmission 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-interval":50,"remote-receive-interval":300,"remote-transmit-
↪interval":300,"remote-echo-interval":50}
```

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
             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}
```

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 a specific IP address 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.

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.

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 BGP protocol uses the AS number 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*.

no router bgp ASN
Destroy a BGP protocol process with the specified ASN.

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
...
```

In the past this feature done differently and the following commands were required to enable the functionality. They are now deprecated.

Deprecated since version 5.0: This command is deprecated and may be safely removed from the config.

bgp multiple-instance
Enable BGP multiple instance feature. Because this is now the default configuration this command will not be displayed in the running configuration.

Deprecated since version 5.0: This command is deprecated and may be safely removed from the config.

no bgp multiple-instance
In previous versions of FRR, this command disabled the BGP multiple instance feature. This functionality is automatically turned on when BGP multiple instances or views exist so this command no longer does anything.

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
 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.

Administrative Distance Metrics

distance bgp (1-255) (1-255) (1-255)
This command change distance value of BGP. The arguments are the distance values for 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

[no] bgp ebgp-requires-policy
This command requires incoming and outgoing filters to be applied for eBGP sessions. Without the incoming filter, no routes will be accepted. Without the outgoing filter, no routes will be announced.

Route Flap Dampening

bgp dampening (1-45) (1-20000) (1-20000) (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.

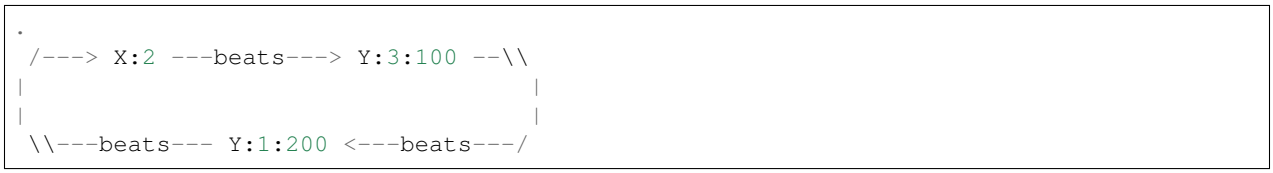
Deterministic local evaluation can *not* fix the second, more major, issue of MED however. Which is that the non-transitive preference of routes MED can cause may lead to routing instability or oscillation across multiple speakers in iBGP topologies. This can occur with full-mesh iBGP, but is particularly 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:

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:



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.

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.

no network A.B.C.D/M

Route Aggregation

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

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. Aggregated routes will not be announce.

no aggregate-address A.B.C.D/M

Redistribution

redistribute kernel
Redistribute kernel route to BGP process.

redistribute static
Redistribute static route to BGP process.

redistribute connected
Redistribute connected route to BGP process.

redistribute rip
Redistribute RIP route to BGP process.

redistribute ospf
Redistribute OSPF route to BGP process.

redistribute vnc
Redistribute VNC routes to BGP process.

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

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 BGP process is cleared using 'clear ip bgp *'. 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.

[no] 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.

Configuring Peers

[no] neighbor PEER shutdown
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.

[no] neighbor PEER disable-connected-check
Allow peerings between directly connected eBGP peers using loopback addresses.

[no] neighbor PEER ebgp-multihop

[no] neighbor PEER description ...
Set description of the peer.

[no] neighbor PEER version VERSION
Set up the neighbor's BGP version. *version* can be *4*, *4+* or *4-*. BGP version *4* is the default value used for BGP peering. BGP version *4+* means that the neighbor supports Multiprotocol Extensions for BGP-4. BGP version *4-* is similar but the neighbor speaks the old Internet-Draft revision 00's Multiprotocol Extensions for BGP-4. Some routing software is still using this version.

[no] 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.

This command is deprecated and may be removed in a future release. Its use should be avoided.

[no] neighbor PEER next-hop-self [all]
This command specifies an announced route's nexthop as being equivalent to the address of the bgp router if it is learned via eBGP. If the optional keyword *all* is specified the modification is done also for routes learned via iBGP.

[no] 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
```

[no] neighbor PEER default-originate
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.

neighbor PEER port PORT

neighbor PEER send-community

[no] neighbor PEER weight WEIGHT
This command specifies a default *weight* value for the neighbor's routes.

[no] neighbor PEER maximum-prefix NUMBER
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.

[no] 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.

[no] 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*.

[no] 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.

[no] bgp fast-external-failover
This command causes bgp to not 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.

[no] bgp default ipv4-unicast
This command allows the user to specify that v4 peering 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.

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.

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.

Capability Negotiation

neighbor PEER strict-capability-match

no 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

no 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.

neighbor PEER override-capability

no neighbor PEER override-capability
Override the result of Capability Negotiation with local configuration. Ignore remote peer's capability value.

AS Path Access Lists

AS path access list is user defined AS path.

ip as-path access-list WORD permit|deny LINE
This command defines a new AS path access list.

no ip as-path access-list WORD

no ip as-path access-list WORD permit|deny LINE

Using AS Path in Route Map

[no] 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.

[no] 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.

[no] 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.

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.

internet `internet` represents well-known communities value 0.

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. It is recommended upon receiving prefixes tagged with this community to add `NO_EXPORT` and `NO_ADVERTISE`.

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.

ip 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.

ip 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.

ip 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.

no ip community-list [standard|expanded] NAME
Deletes the community list specified by NAME. All community lists share the same namespace, so it's not necessary to specify standard or expanded; these modifiers are purely aesthetic.

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

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

  # show ip community-list CLIST
  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 and 99 is standard community list. Community list number in the range from 100 to 199 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.

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

ip community-list (100-199) permit|deny COMMUNITY
This command defines a new expanded community list. The argument to (100-199) defines the list identifier.

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 ollowing commands can be used in route maps:

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
!
ip community-list 70 permit 7675:70
ip community-list 70 deny
ip community-list 80 permit 7675:80
ip community-list 80 deny
ip community-list 90 permit 7675:90
ip community-list 90 deny
!
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` or `0:90`. 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
!
ip community-list 1 permit 0:80 0:90
!
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
!
ip community-list standard FILTER deny 1:1
ip community-list standard FILTER permit
!
route-map RMAP permit 10
 match community FILTER
```

The communities value keyword `internet` has special meanings in standard community lists. In the below example `internet` matches all BGP routes even if the route does not have communities attribute at all. So community list `INTERNET` is the same as `FILTER` in the previous example.

```
ip community-list standard INTERNET deny 1:1
ip community-list standard INTERNET permit internet
```

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
!
ip 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

ip 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.

ip 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.

no ip extcommunity-list NAME

no ip extcommunity-list standard NAME

no ip extcommunity-list expanded NAME
These commands delete extended community lists specified by *name*. All of extended community lists shares a single name space. So extended community lists can be removed simply specifying the name.

show ip extcommunity-list

show ip extcommunity-list NAME
This command displays current extcommunity-list information. When *name* is specified the community list's information is shown.:

```
# show ip extcommunity-list
```

BGP Extended Communities in Route Map

match extcommunity WORD

set extcommunity rt EXTCOMMUNITY
This command set Route Target value.

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

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*.

ip 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.

ip 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.

no ip large-community-list NAME

no ip large-community-list standard NAME

no ip large-community-list expanded NAME
These commands delete Large Community lists specified by *name*. All Large Community lists share a single namespace. This means Large Community lists can be removed by simply specifying the name.

show ip large-community-list

show ip large-community-list NAME
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
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.

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.

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.

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 intemediary. 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.

no rd vpn export [AS:NN|IP:nn]
Deletes any previously-configured export route distinguisher.

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*.

no rt vpn import|export|both [RTLIST...]
Deletes any previously-configured import or export route-target list.

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

no label vpn export [(0..1048575)|auto]
Deletes any previously-configured export label.

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).

no nexthop vpn export [A.B.C.D|X:X::X:X]
Deletes any previously-configured export nexthop.

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.

no route-map vpn import|export [MAP]
Deletes any previously-configured import or export route-map.

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

no import|export vpn
Disables 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.

no import vrf VRFNAME
Disables automatic leaking from vrf VRFNAME to the current VRF using the VPN RIB as intermediary.

Cisco Compatibility

FRR has commands that change some configuration syntax and default behavior to behave more closely to Cisco conventions. These are deprecated and will be removed in a future version of FRR.

Deprecated since version 5.0: Please transition to using the FRR specific syntax for your configuration.

bgp config-type cisco
Cisco compatible BGP configuration output.

When this configuration line is specified:

- `no synchronization` is displayed. This command does nothing and is for display purposes only.
- `no auto-summary` is displayed.
- The `network` and `aggregate-address` arguments are displayed as:

```
A.B.C.D M.M.M.M

FRR: network 10.0.0.0/8
Cisco: network 10.0.0.0

FRR: aggregate-address 192.168.0.0/24
Cisco: aggregate-address 192.168.0.0 255.255.255.0
```

Community attribute handling is also different. If no configuration is specified community attribute and extended community attribute are sent to the neighbor. If a user manually disables the feature, the community attribute is not sent to the neighbor. When `bgp config-type cisco` is specified, the community attribute is not sent to the neighbor by default. To send the community attribute user has to specify `neighbor A.B.C.D send-community` like so:

```
!
router bgp 1
 neighbor 10.0.0.1 remote-as 1
 address-family ipv4 unicast
  no neighbor 10.0.0.1 send-community
 exit-address-family
!
router bgp 1
 neighbor 10.0.0.1 remote-as 1
 address-family ipv4 unicast
  neighbor 10.0.0.1 send-community
 exit-address-family
!
```

Deprecated since version 5.0: Please transition to using the FRR specific syntax for your configuration.

bgp config-type zebra
FRR style BGP configuration. This is the default.

Debugging

show debug
Show all enabled debugs.

[no] 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.

[no] 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.

[no] 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.

[no] debug bgp bestpath <A.B.C.D/M|X:X::X:X/M>
Enable or disable debugging for bestpath selection on the specified prefix.

[no] debug bgp nht
Enable or disable debugging of BGP nexthop tracking.

[no] debug bgp update-groups
Enable or disable debugging of dynamic update groups. This provides general information on group creation, deletion, join and prune events.

[no] 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]

no dump bgp all [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]

no dump bgp updates [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

no dump bgp route-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

clear bgp ipv4|ipv6 *
Clear all address family peers.

clear bgp ipv4|ipv6 PEER
Clear peers which have addresses of X.X.X.X

clear bgp ipv4|ipv6 PEER soft in
Clear peer using soft reconfiguration.

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

show ip bgp A.B.C.D

show bgp

show bgp X:X::X:X
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
```

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 summary
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]`.

show bgp [afi] [safi]

show bgp <ipv4|ipv6> <unicast|multicast|vpn|labeled-unicast>
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 [afi] [safi] summary
Show a bgp peer summary for the specified address family, and subsequent address-family.

show bgp [afi] [safi] neighbor [PEER]
This command shows information on a specific BGP peer of the relevant afi and safi selected.

show bgp [afi] [safi] dampening dampened-paths
Display paths suppressed due to dampening of the selected afi and safi selected.

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

Displaying Routes by Community Attribute

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

show [ip] bgp <ipv4|ipv6> community

show [ip] bgp <ipv4|ipv6> community COMMUNITY

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

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

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.

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

no 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

3.3.6 Routing Policy

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

```
bgp multiple-instance
!
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.

3.3.7 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 `(^|[,{}()]|$)`.

3.3.8 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
```

(continues on next page)

(continued from previous page)

```
  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

 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
```

(continues on next page)

(continued from previous page)

```
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
!
ip as-path access-list asp-own-as permit ^$
ip as-path access-list 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
ip community-list standard cm-blackhole permit 64512:100
!
! set no-export community before advertising
ip community-list standard cm-set-no-export permit 64512:200
!
! advertise only to other customers
ip community-list standard cm-cust-only permit 64512:300
!
! advertise only to upstreams
ip community-list standard cm-upstream-only permit 64512:400
!
! advertise to upstreams with no-export
ip community-list standard cm-upstream-noexport permit 64512:500
!
! set local-pref to least significant 3 digits of the community
ip community-list standard cm-prefmod-100 permit 64512:2100
ip community-list standard cm-prefmod-200 permit 64512:2200
ip community-list standard cm-prefmod-300 permit 64512:2300
ip community-list standard cm-prefmod-400 permit 64512:2400
ip community-list expanded cme-prefmod-range permit 64512:2...
!
! Informational communities
!
! 3000 - learned from upstream
! 3100 - learned from customer
! 3200 - learned from peer
!
ip community-list standard cm-learnt-upstream permit 64512:3000
```

(continues on next page)

(continued from previous page)

```
ip community-list standard cm-learnt-cust permit 64512:3100
ip 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
 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
```

(continues on next page)

(continued from previous page)

```
 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
 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
```

(continues on next page)

(continued from previous page)

```
 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
!
log file bgpd.log
!
```

3.3.9 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.

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

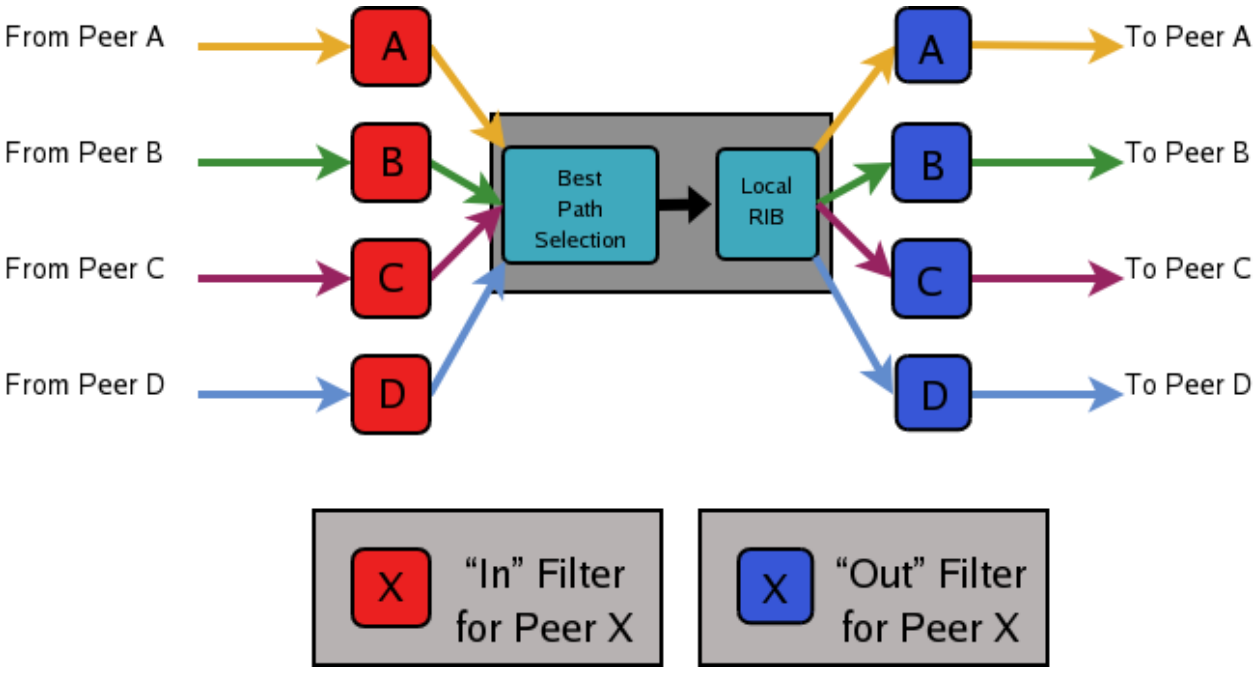


Fig. 1: Announcement processing inside a 'normal' BGP speaker

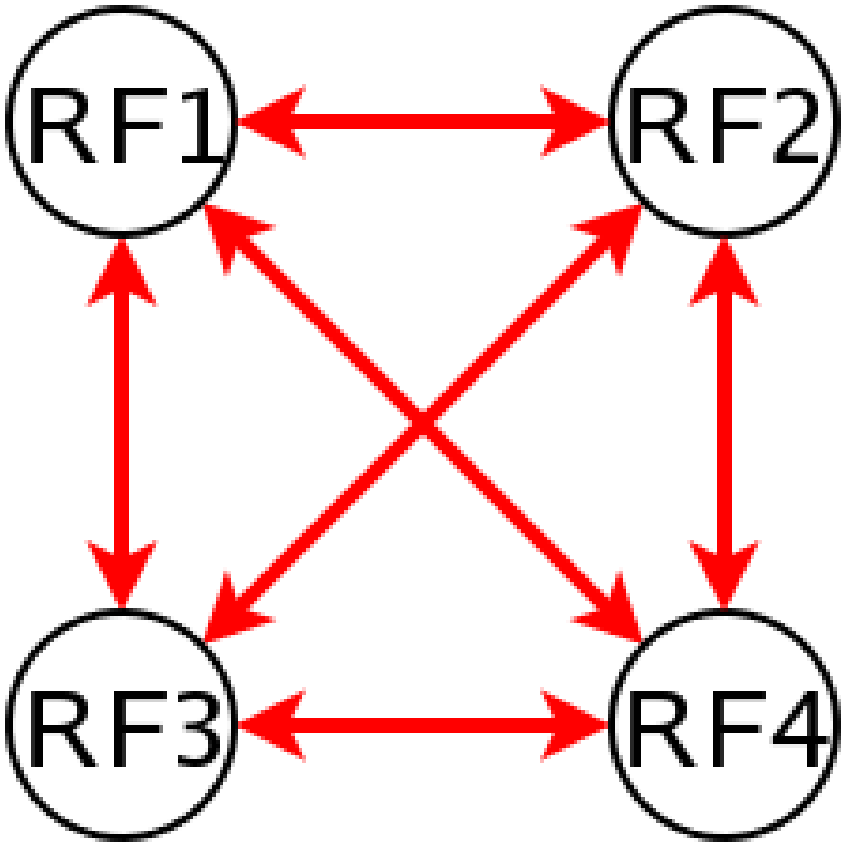


Fig. 2: Full Mesh

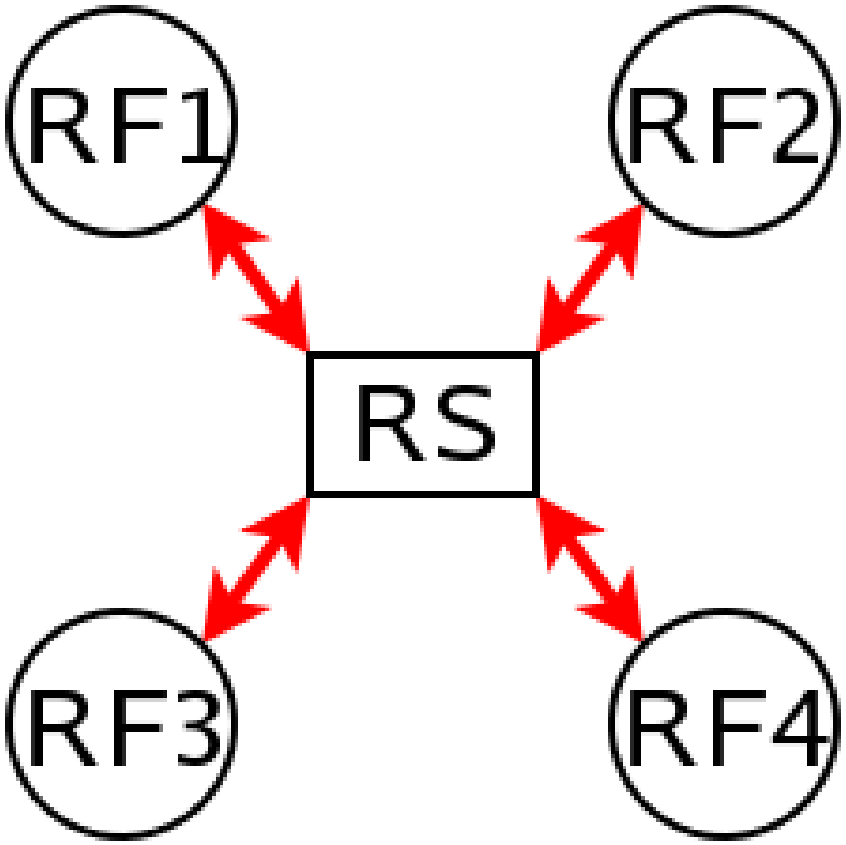


Fig. 3: Route server and clients

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).

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.

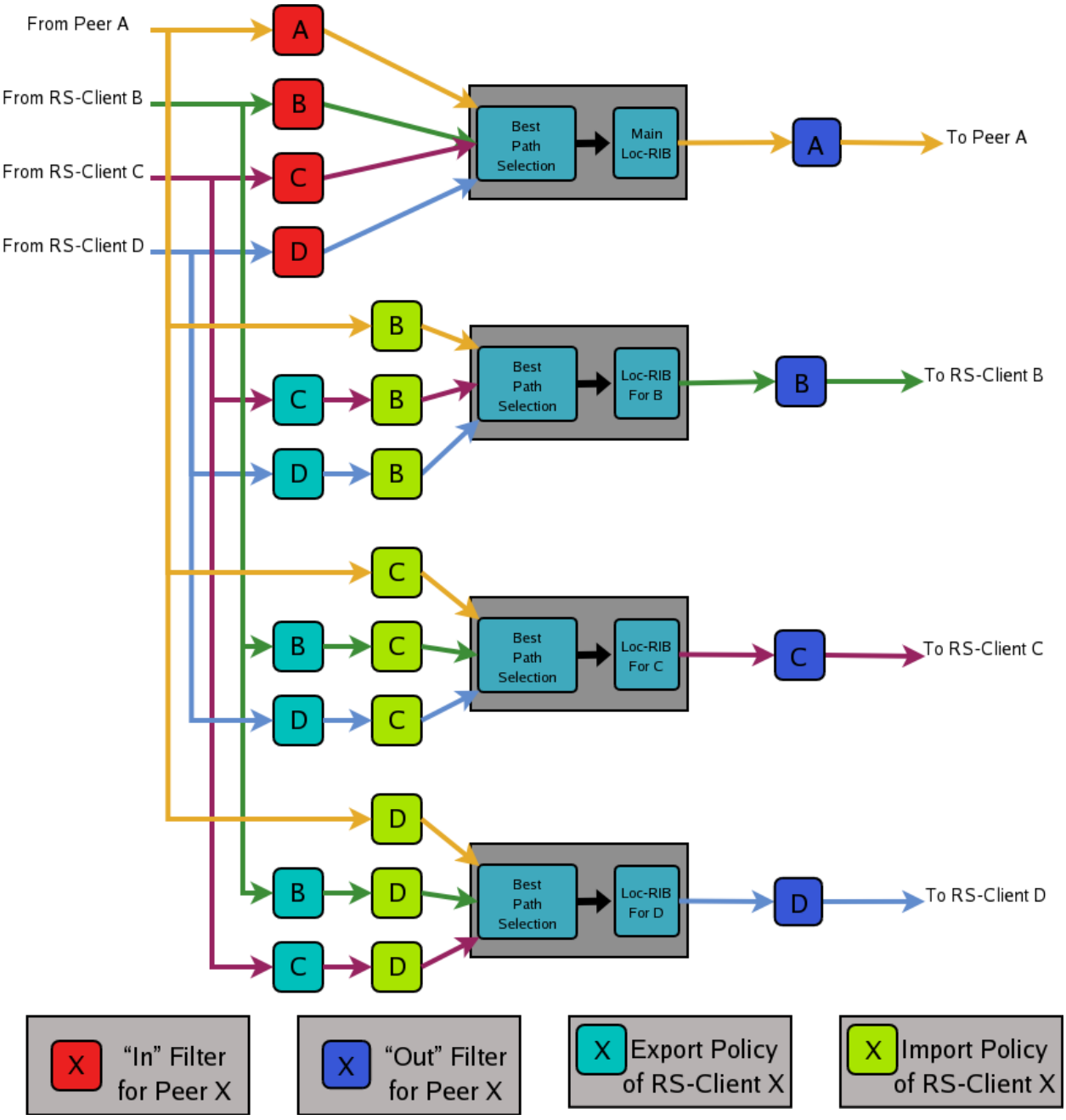


Fig. 4: Announcement processing model implemented by the Route Server

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 import|export
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
```

(continues on next page)

(continued from previous page)

```
  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
  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
!
bgp multiple-instance
!
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 import
    neighbor 2001:0DB8::A route-map RSCLIENT-A-EXPORT export
    neighbor 2001:0DB8::A soft-reconfiguration inbound

    neighbor 2001:0DB8::B activate
    neighbor 2001:0DB8::B route-server-client
```

(continues on next page)

(continued from previous page)

```
    neighbor 2001:0DB8::B route-map RSCLIENT-B-IMPORT import
    neighbor 2001:0DB8::B route-map RSCLIENT-B-EXPORT export
    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 import
    neighbor 2001:0DB8::C route-map RSCLIENT-C-EXPORT export
    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
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 import
...
!
...
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
```

(continues on next page)

(continued from previous page)

```
  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.

3.3.10 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

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.

When first installing FRR with RPKI support from the pre-packaged binaries. 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)

no rpki polling_period
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 timeout <1-4,294,967,296>

no rpki timeout
Set the number of seconds the router waits for the cache reply. If the cache server is not replying within this time period, the router deletes all received prefix records from the prefix table.

The default value is 600 seconds.

rpki initial-synchronisation-timeout <1-4,294,967,296>

no rpki initial-synchronisation-timeout
Set the number of seconds until the first synchronization with the cache server needs to be completed. If the timeout expires, BGP routing is started without RPKI. The router will try to establish the cache server connection in the background.

The default value is 30 seconds.

The following commands configure one or multiple cache servers.

rpki cache (A.B.C.D|WORD) PORT [SSH_USERNAME] [SSH_PRIVKEY_PATH] [SSH_PUBKEY_PATH] [KNOWN_

no rpki cache (A.B.C.D|WORD) [PORT] PREFERENCE
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.

SSH_PUBKEY_PATH Local path that includes the public 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`.

Validating BGP Updates

match rpki notfound|invalid|valid

no match rpki notfound|invalid|valid
Create a clause for a route map to match prefixes with the specified RPKI state.

Note that the matching of invalid prefixes requires that invalid prefixes are considered for best path selection, i.e., `bgp bestpath prefix-validate disallow-invalid` is not enabled.

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
```

Debugging

debug rpki

no debug rpki
Enable or disable debugging output for RPKI.

Displaying RPKI

show rpki prefix-table
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-connection
Display all configured cache servers, whether active or not.

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 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
!
 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
!
```

3.3.11 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.

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]

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>
  address-family ipv4 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]

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.

[no] 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

[no] 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
```

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.

[no] 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`.

[no] 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
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]

3.4 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

[no] router babel
Enable or disable Babel routing.

[no] babel resend-delay (20-655340)
Specifies the time after which important messages are resent when avoiding a black-hole. The default is 2000 ms.

[no] 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.

no 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.

[no] 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 100 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 0, which effectively disables the use of a RTT-based cost.

[no] 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

[no] 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

[no] 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.

3.5 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

no router openfabric WORD
Enable or disable the OpenFabric process by specifying the OpenFabric domain with 'WORD'.

net XX.XXXX. ... .XXX.XX

no net XX.XXXX. ... .XXX.XX
Set/Unset network entity title (NET) provided in ISO format.

domain-password [clear | md5] <password>

no domain-password
Configure the authentication password for a domain, as clear text or md5 one.

log-adjacency-changes

no log-adjacency-changes
Log changes in adjacency state.

set-overload-bit

no set-overload-bit
Set overload bit to avoid any transit traffic.

purge-originator

no purge-originator
Enable or disable **RFC 6232** purge originator identification.

fabric-tier (0-14)

no fabric-tier
Configure a static tier number to advertise as location in the fabric

3.5.3 OpenFabric Timer

lsp-gen-interval (1-120)

no lsp-gen-interval
Set minimum interval in seconds between regenerating same LSP.

lsp-refresh-interval (1-65235)

no lsp-refresh-interval
Set LSP refresh interval in seconds.

max-lsp-lifetime (360-65535)

no max-lsp-lifetime
Set LSP maximum LSP lifetime in seconds.

spf-interval (1-120)

no spf-interval
Set minimum interval between consecutive SPF calculations in seconds.

3.5.4 OpenFabric interface

ip router openfabric WORD

no 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)

no openfabric csnp-interval
Set CSNP interval in seconds.

openfabric hello-interval (1-600)

no openfabric hello-interval
Set Hello interval in seconds.

openfabric hello-multiplier (2-100)

no openfabric hello-multiplier
Set multiplier for Hello holding time.

openfabric metric (0-16777215)

no openfabric metric
Set interface metric value.

openfabric passive

no openfabric passive
Configure the passive mode for this interface.

openfabric password [clear | md5] <password>

no openfabric password
Configure the authentication password (clear or encoded text) for the interface.

openfabric psnp-interval (1-120)

no openfabric psnp-interval
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

no debug openfabric adj-packets

OpenFabric Adjacency related packets.

debug openfabric checksum-errors

no debug openfabric checksum-errors

OpenFabric LSP checksum errors.

debug openfabric events

no debug openfabric events

OpenFabric Events.

debug openfabric local-updates

no debug openfabric local-updates

OpenFabric local update packets.

debug openfabric lsp-gen

no debug openfabric lsp-gen

Generation of own LSPs.

debug openfabric lsp-sched

no debug openfabric lsp-sched

Debug scheduling of generation of own LSPs.

debug openfabric packet-dump

no debug openfabric packet-dump

OpenFabric packet dump.

debug openfabric protocol-errors

no debug openfabric protocol-errors

OpenFabric LSP protocol errors.

debug openfabric route-events

no debug openfabric route-events

OpenFabric Route related events.

debug openfabric snp-packets

no debug openfabric snp-packets

OpenFabric CSNP/PSNP packets.

debug openfabric spf-events

debug openfabric spf-statistics

debug openfabric spf-triggers

no debug openfabric spf-events

no debug openfabric spf-statistics

no debug openfabric spf-triggers

OpenFabric Shortest Path First Events, Timing and Statistic Data and triggering events.

debug openfabric update-packets

no debug openfabric update-packets

Update related packets.

show debugging openfabric
Print which OpenFabric debug levels are active.

3.5.7 OpenFabric configuration example

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
!
interface eth1
 ip router openfabric 1
 ipv6 router openfabric 1
!
router openfabric 1
 net 49.0000.0000.0001.00
```

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*).

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

[no] mpls ldp
Enable or disable LDP daemon

[no] router-id A.B.C.D
The following command located under MPLS router node configures the MPLS router-id of the local device.

[no] address-family [ipv4 | ipv6]
Configure LDP for IPv4 or IPv6 address-family. Located under MPLS route node, this subnode permits configuring the LDP neighbors.

[no] 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.

[no] 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.

[no] 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.

[no] 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.

[no] discovery hello holdtime HOLDTIME

[no] 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.

[no] 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

[no] 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 LDP Example 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
```

(continues on next page)

(continued from previous page)

```
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#
```

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/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)`
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.

no router eigrp (1-65535)
Disable EIGRP.

network NETWORK

no 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)

no passive-interface IFNAME
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 kernel

redistribute kernel metric (1-4294967295) (0-4294967295) (0-255) (1-255) (1-65535)

no redistribute kernel
redistribute kernel redistributes routing information from kernel route entries into the EIGRP tables. *no redistribute kernel* disables the routes.

redistribute static

redistribute static metric (1-4294967295) (0-4294967295) (0-255) (1-255) (1-65535)

no redistribute static
redistribute static redistributes routing information from static route entries into the EIGRP tables. *no redistribute static* disables the routes.

redistribute connected

redistribute connected metric (1-4294967295) (0-4294967295) (0-255) (1-255) (1-65535)

no redistribute connected
Redistribute connected routes into the EIGRP tables. *no redistribute connected* disables the connected routes in the EIGRP tables. This command redistribute connected of the interface which EIGRP disabled. The connected route on EIGRP enabled interface is announced by default.

redistribute ospf

redistribute ospf metric (1-4294967295) (0-4294967295) (0-255) (1-255) (1-65535)

no redistribute ospf
redistribute ospf redistributes routing information from ospf route entries into the EIGRP tables. *no redistribute ospf* disables the routes.

redistribute bgp

redistribute bgp metric (1-4294967295) (0-4294967295) (0-255) (1-255) (1-65535)

no redistribute bgp
redistribute bgp redistributes routing information from bgp route entries into the EIGRP tables. *no redistribute bgp* disables the routes.

3.7.4 Show EIGRP Information

show ip eigrp 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
```

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.

3.8 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.

3.8.1 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`.

3.8.2 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.

[no] router isis WORD
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

no net XX.XXXX. ... .XXX.XX
Set/Unset network entity title (NET) provided in ISO format.

hostname dynamic

no hostname dynamic
Enable support for dynamic hostname.

area-password [clear | md5] <password>

domain-password [clear | md5] <password>

no area-password

no domain-password
Configure the authentication password for an area, respectively a domain, as clear text or md5 one.

log-adjacency-changes

no log-adjacency-changes
Log changes in adjacency state.

metric-style [narrow | transition | wide]

no metric-style
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

set-overload-bit

no set-overload-bit
Set overload bit to avoid any transit traffic.

purge-originator

no purge-originator
Enable or disable **RFC 6232** purge originator identification.

3.8.3 ISIS Timer

lsp-gen-interval (1-120)

lsp-gen-interval [level-1 | level-2] (1-120)

no lsp-gen-interval

no lsp-gen-interval [level-1 | level-2]
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)

no lsp-refresh-interval [level-1 | level-2]
Set LSP refresh interval in seconds, globally, for an area (level-1) or a domain (level-2).

max-lsp-lifetime (360-65535)

max-lsp-lifetime [level-1 | level-2] (360-65535)

no max-lsp-lifetime

no max-lsp-lifetime [level-1 | level-2]
Set LSP maximum LSP lifetime in seconds, globally, for an area (level-1) or a domain (level-2).

spf-interval (1-120)

spf-interval [level-1 | level-2] (1-120)

no spf-interval

no spf-interval [level-1 | level-2]
Set minimum interval between consecutive SPF calculations in seconds.

3.8.4 ISIS region

is-type [level-1 | level-1-2 | level-2-only]

no is-type
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

3.8.5 ISIS interface

[no] <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]

no isis circuit-type
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)

isis csnp-interval (1-600) [level-1 | level-2]

no isis csnp-interval

no isis csnp-interval [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-interval (1-600)

isis hello-interval (1-600) [level-1 | level-2]

no isis hello-interval

no isis hello-interval [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)

isis hello-multiplier (2-100) [level-1 | level-2]

no isis hello-multiplier

no isis hello-multiplier [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)]

isis metric [(0-255) | (0-16777215)] [level-1 | level-2]

no isis metric

no isis metric [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

no isis network point-to-point
Set network type to 'Point-to-Point' (broadcast by default).

isis passive

no isis passive
Configure the passive mode for this interface.

isis password [clear | md5] <password>

no isis password
Configure the authentication password (clear or encoded text) for the interface.

isis priority (0-127)

isis priority (0-127) [level-1 | level-2]

no isis priority

no isis priority [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)

isis psnp-interval (1-120) [level-1 | level-2]

no isis psnp-interval

no isis psnp-interval [level-1 | level-2]
Set PSNP interval in seconds globally, for an area (level-1) or a domain (level-2).

isis three-way-handshake

no isis three-way-handshake
Enable or disable **RFC 5303** Three-Way Handshake for P2P adjacencies. Three-Way Handshake is enabled by default.

3.8.6 Showing ISIS information

show isis summary
Show summary information about ISIS.

show isis hostname
Show information about ISIS node.

show isis interface

show isis interface detail

show isis interface <interface name>
Show state and configuration of ISIS specified interface, or all interfaces if no interface is given with or without details.

show isis neighbor

show isis neighbor <System Id>

show isis neighbor detail
Show state and information of ISIS specified neighbor, or all neighbors if no system id is given with or without details.

show isis database

show isis database [detail]

show isis database <LSP id> [detail]

show isis database detail <LSP id>
Show the ISIS database globally, for a specific LSP id without or with details.

show isis topology

show isis topology [level-1|level-2]
Show topology IS-IS paths to Intermediate Systems, globally, in area (level-1) or domain (level-2).

show ip route isis
Show the ISIS routing table, as determined by the most recent SPF calculation.

3.8.7 Traffic Engineering

mpls-te on

no mpls-te
Enable Traffic Engineering LSP flooding.

mpls-te router-address <A.B.C.D>

no mpls-te router-address
Configure stable IP address for MPLS-TE.

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.

See also:

Traffic Engineering

3.8.8 Debugging ISIS

debug isis adj-packets

no debug isis adj-packets
IS-IS Adjacency related packets.

debug isis checksum-errors

no debug isis checksum-errors
IS-IS LSP checksum errors.

debug isis events

no debug isis events
IS-IS Events.

debug isis local-updates

no debug isis local-updates
IS-IS local update packets.

debug isis packet-dump

no debug isis packet-dump
IS-IS packet dump.

debug isis protocol-errors

no debug isis protocol-errors
IS-IS LSP protocol errors.

debug isis route-events

no debug isis route-events
IS-IS Route related events.

debug isis snp-packets

no debug isis snp-packets
IS-IS CSNP/PSNP packets.

debug isis spf-events

debug isis spf-statistics

debug isis spf-triggers

no debug isis spf-events

no debug isis spf-statistics

no debug isis spf-triggers
IS-IS Shortest Path First Events, Timing and Statistic Data and triggering events.

debug isis update-packets

no debug isis update-packets
Update related packets.

show debugging isis
Print which ISIS debug level is activate.

3.8.9 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
```

(continues on next page)

(continued from previous page)

```
  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 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
```

3.9 NHRP

nhrpd is an implementation of the :abbr: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.

Cisco Dynamic Multipoint VPN (DMVPN) is based on NHRP, and frr nhrpd implements this scenario.

3.9.1 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
```

3.9.2 Configuring NHRP

FIXME

3.9.3 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
```

To start sending these traffic notices out from hubs, use the nhrp per-interface directive:

```
interface gre1
 ip nhrp redirect
```

3.9.4 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 (hardcoded now as /var/run/charon.vici).

strongSwan currently needs few patches applied. Please check out the http://git.alpinelinux.org/cgit/user/tteras/strongswan/log/?h=tteras-release,release and http://git.alpinelinux.org/cgit/user/tteras/strongswan/log/?h=tteras,working tree git repositories for the patches.

3.9.5 NHRP Events

FIXME

3.9.6 Configuration Example

FIXME

3.10 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.

3.10.1 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 Edgar Djikstra'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 greater detail than here.

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 `passive-interface INTERFACE`.

- 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

    Link connected to: Stub Network
     (Link ID) Net: 192.168.3.190
     (Link Data) Network Mask: 255.255.255.255
```

(continues on next page)

(continued from previous page)

```
      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
  |         |           |
  |         |     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
  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.

3.10.2 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
   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

no router ospf [(1-65535)] vrf NAME
Enable or disable the OSPF process.

ospf router-id A.B.C.D

no 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

no 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

no 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]

no 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 INTERFACE

no passive-interface INTERFACE
Do not speak OSPF interface on the given 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.

timers throttle spf DELAY INITIAL-HOLDTIME MAX-HOLDTIME

no timers throttle spf
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|on-shutdown] (5-86400)

max-metric router-lsa administrative

no max-metric router-lsa [on-startup|on-shutdown|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)

no auto-cost reference-bandwidth

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)

no network A.B.C.D/M area A.B.C.D

no 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.

In some cases it may be more convenient to enable OSPF on a per interface/subnet basis (`ip ospf area AREA [ADDR]`).

Areas

area A.B.C.D range A.B.C.D/M

area (0-4294967295) range A.B.C.D/M

no area A.B.C.D range A.B.C.D/M

no area (0-4294967295) range A.B.C.D/M

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. Summarizing Type-7 AS-external-LSAs isn't supported yet by FRR.

```
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 IPV4_PREFIX not-advertise

no area A.B.C.D range IPV4_PREFIX 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 IPV4_PREFIX substitute IPV4_PREFIX

no area A.B.C.D range IPV4_PREFIX substitute IPV4_PREFIX

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. 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

no area A.B.C.D virtual-link A.B.C.D

no area (0-4294967295) virtual-link A.B.C.D

area A.B.C.D shortcut

area (0-4294967295) shortcut

no area A.B.C.D shortcut

no 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

no area A.B.C.D stub

no 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

no area A.B.C.D stub no-summary

no 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 default-cost (0-16777215)

no 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

no area A.B.C.D export-list NAME

no 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

no area A.B.C.D import-list NAME

no 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

no area A.B.C.D filter-list prefix NAME in

no area A.B.C.D filter-list prefix NAME out

no area (0-4294967295) filter-list prefix NAME in

no 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

no area A.B.C.D authentication

no 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]

no ip ospf area [ADDR]
Enable OSPF on the interface, optionally restricted to just the IP address given by *ADDR*, putting it in the *AREA* area. Per interface area settings take precedence to network commands (`network A.B.C.D/M area A.B.C.D`).

If you have a lot of interfaces, and/or a lot of subnets, then enabling OSPF via this command may result in a slight performance improvement.

ip ospf authentication-key AUTH_KEY

no ip ospf authentication-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

no ip ospf message-digest-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)

no ip ospf cost
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)

no ip ospf dead-interval
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)

no ip ospf hello-interval
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 network (broadcast|non-broadcast|point-to-multipoint|point-to-point)

no ip ospf network
Set explicitly network type for specified interface.

ip ospf priority (0-255)

no ip ospf priority
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)

no ip ospf retransmit interval
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

no ip ospf transmit-delay
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 area (A.B.C.D|(0-4294967295))

no ip ospf area
Enable ospf on an interface and set associated area.

Redistribution

redistribute (kernel|connected|static|rip|bgp)

redistribute (kernel|connected|static|rip|bgp) ROUTE-MAP

redistribute (kernel|connected|static|rip|bgp) metric-type (1|2)

redistribute (kernel|connected|static|rip|bgp) metric-type (1|2) route-map WORD

redistribute (kernel|connected|static|rip|bgp) metric (0-16777214)

redistribute (kernel|connected|static|rip|bgp) metric (0-16777214) route-map WORD

redistribute (kernel|connected|static|rip|bgp) metric-type (1|2) metric (0-16777214)

redistribute (kernel|connected|static|rip|bgp) metric-type (1|2) metric (0-16777214) route-

no redistribute (kernel|connected|static|rip|bgp)

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 *passive-interface* configuration.

See also:

clicmd:*passive-interface INTERFACE*.

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

no default-information originate

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|ospf

no distribute-list NAME out (kernel|connected|static|rip|ospf

Apply the access-list filter, NAME, to redistributed routes of the given type before allowing the routes to redistributed into OSPF (*ospf redistribution*).

default-metric (0-16777214)

no default-metric

distance (1-255)

no distance (1-255)

distance ospf (intra-area|inter-area|external) (1-255)

no distance ospf

router zebra

no router zebra

3.10.3 Showing Information

show ip ospf
Show information on a variety of general OSPF and area state and configuration information.

show ip ospf interface [INTERFACE]
Show state and configuration of OSPF the specified interface, or all interfaces if no interface is given.

show ip ospf neighbor

show ip ospf neighbor INTERFACE

show ip ospf neighbor detail

show ip ospf neighbor INTERFACE detail

show ip ospf database

show ip ospf database (asbr-summary|external|network|router|summary)

show ip ospf database (asbr-summary|external|network|router|summary) LINK-STATE-ID

show ip ospf database (asbr-summary|external|network|router|summary) LINK-STATE-ID adv-rout

show ip ospf database (asbr-summary|external|network|router|summary) adv-router ADV-ROUTER

show ip ospf database (asbr-summary|external|network|router|summary) LINK-STATE-ID self-ori

show ip ospf database (asbr-summary|external|network|router|summary) self-originate

show ip ospf database max-age

show ip ospf database self-originate

show ip ospf route
Show the OSPF routing table, as determined by the most recent SPF calculation.

3.10.4 Opaque LSA

ospf opaque-lsa

capability opaque

no ospf opaque-lsa

no capability opaque
ospfd supports Opaque LSA (**RFC 2370**) as fundamental for MPLS Traffic Engineering LSA. Prior to used MPLS TE, opaque-lsa must be enable in the configuration file. Alternate command could be "mpls-te on" (*Traffic Engineering*).

show ip ospf database (opaque-link|opaque-area|opaque-external)

show ip ospf database (opaque-link|opaque-area|opaque-external) LINK-STATE-ID

show ip ospf database (opaque-link|opaque-area|opaque-external) LINK-STATE-ID adv-router AD

show ip ospf database (opaque-link|opaque-area|opaque-external) adv-router ADV-ROUTER

show ip ospf database (opaque-link|opaque-area|opaque-external) LINK-STATE-ID self-originat

show ip ospf database (opaque-link|opaque-area|opaque-external) self-originate
Show Opaque LSA from the database.

3.10.5 Traffic Engineering

mpls-te on

no mpls-te
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

no mpls-te inter-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.

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.

3.10.6 Router Information

router-info [as | area]

no router-info
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>

no pce address

pce domain as (0-65535)

no pce domain as (0-65535)

pce neighbor as (0-65535)

no pce neighbor as (0-65535)

pce flag BITPATTERN

no pce flag

pce scope BITPATTERN

no pce scope
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.

3.10.7 Segment Routing

This is an EXPERIMENTAL support of Segment Routing as per draft *draft-ietf-ospf-segment-routing-extensions-24.txt* for MPLS dataplane.

[no] 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.

[no] segment-routing global-block (0-1048575) (0-1048575)
Fix the Segment Routing Global Block i.e. the label range used by MPLS to store label in the MPLS FIB.

[no] 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.

[no] segment-routing prefix A.B.C.D/M index (0-65535) [no-php-flag]
Set the Segment Routing index for the specified prefix. Note that, only 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.

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.

3.10.8 Debugging OSPF

debug ospf packet (hello|dd|ls-request|ls-update|ls-ack|all) (send|recv) [detail]

no debug ospf packet (hello|dd|ls-request|ls-update|ls-ack|all) (send|recv) [detail]
Dump Packet for debugging

debug ospf ism

debug ospf ism (status|events|timers)

no debug ospf ism

no debug ospf ism (status|events|timers)
Show debug information of Interface State Machine

debug ospf nsm

debug ospf nsm (status|events|timers)

no debug ospf nsm

no debug ospf nsm (status|events|timers)
Show debug information of Network State Machine

debug ospf event

no debug ospf event
Show debug information of OSPF event

debug ospf nssa

no debug ospf nssa
Show debug information about Not So Stub Area

debug ospf lsa

debug ospf lsa (generate|flooding|refresh)

no debug ospf lsa

no debug ospf lsa (generate|flooding|refresh)
Show debug detail of Link State messages

debug ospf te

no debug ospf te
Show debug information about Traffic Engineering LSA

debug ospf zebra

debug ospf zebra (interface|redistribute)

no debug ospf zebra

no debug ospf zebra (interface|redistribute)
Show debug information of ZEBRA API

show debugging ospf

3.10.9 OSPF Configuration Examples

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
!
interface br0
 ip ospf authentication message-digest
 ip ospf message-digest-key 2 md5 XYZ12345
!
router ospf
 ospf router-id 192.168.0.1
```

(continues on next page)

(continued from previous page)

```
 redistribute connected
 passive interface ppp0
 network 192.168.0.0/24 area 0.0.0.0
 network 10.0.0.0/16 area 0.0.0.0
 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
 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
!
```

3.11 OSPFv3

ospf6d is a daemon support OSPF version 3 for IPv6 network. OSPF for IPv6 is described in **RFC 2740**.

3.11.1 OSPF6 router

router ospf6

ospf6 router-id A.B.C.D
Set router's Router-ID.

interface IFNAME area AREA
Bind interface to specified area, and start sending OSPF packets. *area* can be specified as 0.

timers throttle spf DELAY INITIAL-HOLDTIME MAX-HOLDTIME

no timers throttle spf
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

no auto-cost reference-bandwidth
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.

3.11.2 OSPF6 area

Area support for OSPFv3 is not yet implemented.

3.11.3 OSPF6 interface

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 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.

3.11.4 Redistribute routes to OSPF6

redistribute static

redistribute connected

redistribute ripng

3.11.5 Showing OSPF6 information

show ipv6 ospf6 [INSTANCE_ID]
INSTANCE_ID is an optional OSPF instance ID. To see router ID and OSPF instance ID, simply type "show ipv6 ospf6 <cr>".

show ipv6 ospf6 database
This command shows LSA database summary. You can specify the type of LSA.

show ipv6 ospf6 interface
To see OSPF interface configuration like costs.

show ipv6 ospf6 neighbor
Shows state and chosen (Backup) DR of neighbor.

show ipv6 ospf6 request-list A.B.C.D
Shows requestlist of neighbor.

show ipv6 route ospf6
This command shows internal routing table.

show ipv6 ospf6 zebra
Shows state about what is being redistributed between zebra and OSPF6

3.11.6 OSPF6 Configuration Examples

Example of ospf6d configured on one interface and area:

```
interface eth0
 ipv6 ospf6 instance-id 0
!
router ospf6
 router-id 212.17.55.53
 area 0.0.0.0 range 2001:770:105:2::/64
 interface eth0 area 0.0.0.0
!
```

3.12 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.

3.12.1 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/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 spt-switchover infinity-and-beyond
On the last hop router if it is desired to not switch over to the SPT tree. Configure this command. 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 (60-600)
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.

ip pim keep-alive-timer (31-60000)
Modify the time out value for a S,G flow from 31-60000 seconds. 31 seconds is chosen for a lower bound because 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 (1-100)
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 (5-60000)
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.

3.12.2 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 bfd
Turns on BFD support for PIM for 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 (1-180) (1-180)
Set the pim hello and hold interval for a interface.

ip pim sm
Tell pim that we would like to use this interface to form pim neighbors over. Please note we will *not* accept igmp reports over this interface with this command.

ip igmp
Tell pim to receive IGMP reports and Query on this interface. The default version is v3. This command is useful on the LHR.

ip igmp join A.B.C.D A.B.C.D
Join multicast source-group on an interface.

ip igmp query-interval (1-1800)
Set the IGMP query interval that PIM will use.

ip igmp query-max-response-time (10-250)
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.

3.12.3 PIM Multicast RIB insertion:

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.

3.12.4 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 join
Display IGMP static join information.

show ip igmp groups
Display IGMP groups information.

show ip igmp groups retransmissions
Display IGMP group retransmission information.

show ip igmp sources
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 count
Display information about installed into the kernel S,G mroutes and in addition display data about packet flow for the mroutes.

show ip pim assert
Display information about asserts in the PIM system for S,G mroutes.

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 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 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 rp-info
Display information about RP's that are configured on this router.

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 upstream
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 rpf
Display the multicast RIB created in zebra.

mtrace A.B.C.D [A.B.C.D]
Display multicast traceroute towards source, optionally for particular group.

3.12.5 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 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 zebra
This gathers data about events from zebra that come up through the ZAPI.

3.12.6 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 pim interfaces
Reset PIM interfaces.

clear ip pim oil
Rescan PIM OIL (output interface list).

3.12.7 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.

3.13 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.

3.13.1 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.

3.13.2 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.

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] [nexthop-vrf NAME]
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.

[no] 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

Showing Nexthop Group Information

show pbr nexthop-groups [NAME]
Display information on a PBR nexthop-group. If `NAME` is omitted, all nexthop groups are shown.

3.13.3 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.

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.

3.13.4 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.

3.13.5 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.

3.14 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.

3.14.1 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/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.

3.14.2 RIP Configuration

router rip
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.

no router rip
Disable RIP.

network NETWORK

no 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

no 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

no neighbor A.B.C.D
Specify RIP neighbor. When a neighbor doesn't understand multicast, this command is used to specify neighbors. In some cases, not all routers will be able to understand multicasting, where packets are sent to a network or a group of addresses. In a situation where a neighbor cannot process multicast packets, it is necessary to establish a direct link between routers. The neighbor command allows the network administrator to specify a router as a RIP neighbor. 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)

no passive-interface IFNAME
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

no ip split-horizon
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*.

3.14.3 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 1.

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.

no version
Reset the global version setting back to the default.

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).

3.14.4 How to Announce RIP route

redistribute kernel

redistribute kernel metric (0-16)

redistribute kernel route-map ROUTE-MAP

no redistribute kernel
redistribute kernel redistributes routing information from kernel route entries into the RIP tables. *no redistribute kernel* disables the routes.

redistribute static

redistribute static metric (0-16)

redistribute static route-map ROUTE-MAP

no redistribute static
redistribute static redistributes routing information from static route entries into the RIP tables. *no redistribute static* disables the routes.

redistribute connected

redistribute connected metric (0-16)

redistribute connected route-map ROUTE-MAP

no redistribute connected
Redistribute connected routes into the RIP tables. *no redistribute connected* disables the connected routes in the RIP tables. This command redistribute connected of the interface which RIP disabled. The connected route on RIP enabled interface is announced by default.

redistribute ospf

redistribute ospf metric (0-16)

redistribute ospf route-map ROUTE-MAP

no redistribute ospf
redistribute ospf redistributes routing information from ospf route entries into the RIP tables. *no redistribute ospf* disables the routes.

redistribute bgp

redistribute bgp metric (0-16)

redistribute bgp route-map ROUTE-MAP

no redistribute bgp
redistribute bgp redistributes routing information from bgp route entries into the RIP tables. *no redistribute bgp* disables the routes.

If you want to specify RIP only static routes:

default-information originate

route A.B.C.D/M

no 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*.

3.14.5 Filtering RIP Routes

RIP routes can be filtered by a distribute-list.

distribute-list ACCESS_LIST DIRECT IFNAME
You can apply access lists to the interface with a *distribute-list* command. ACCESS_LIST is the access list name. DIRECT is `in` or `out`. If DIRECT is `in` the access list is applied to input packets.

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.

distribute-list prefix PREFIX_LIST (in|out) IFNAME
You can apply prefix lists to the interface with a *distribute-list* command. PREFIX_LIST is the prefix list name. Next is the direction of `in` or `out`. If DIRECT is `in` the access list is applied to input packets.

3.14.6 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)

no 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

3.14.7 RIP distance

Distance value is used in zebra daemon. Default RIP distance is 120.

distance (1-255)

no distance (1-255)
Set default RIP distance to specified value.

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

no 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

no 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.

3.14.8 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.

3.14.9 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

no ip rip authentication mode md5
Set the interface with RIPv2 MD5 authentication.

ip rip authentication mode text

no ip rip authentication mode text
Set the interface with RIPv2 simple password authentication.

ip rip authentication string STRING

no 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

no 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
!
```

3.14.10 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.

no timers basic
The *no timers basic* command will reset the timers to the default settings listed above.

3.14.11 Show RIP Information

To display RIP routes.

show ip rip
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 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
```

3.14.12 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.

3.15 RIPng

ripngd supports the RIPng protocol as described in **RFC 2080**. It's an IPv6 reincarnation of the RIP protocol.

3.15.1 Invoking ripngd

There are no *ripngd* specific invocation options. Common options can be specified (*Common Invocation Options*).

3.15.2 ripngd Configuration

Currently ripngd supports the following commands:

router ripng
Enable RIPng.

flush_timer TIME
Set flush timer.

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.

router zebra
This command is the default and does not appear in the configuration. With this statement, RIPng routes go to the *zebra* daemon.

3.15.3 ripngd Terminal Mode Commands

show ip ripng

show debugging ripng

debug ripng events

debug ripng packet

debug ripng zebra

3.15.4 ripngd Filtering Commands

distribute-list ACCESS_LIST (in|out) IFNAME
You can apply an access-list to the interface using the *distribute-list* command. ACCESS_LIST is an access-list name. *direct* is `in` or `out`. If *direct* is `in`, the access-list is applied only to incoming packets.:

```
distribute-list local-only out sit1
```

3.16 SHARP

SHARP (Super Happy Advanced Routing Process) is a daemon that provides miscellaneous functionality used for testing FRR and creating proof-of-concept labs.

3.16.1 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.

3.16.2 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) [
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.

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.

[no] sharp watch <nexthop|import> <A.B.C.D|X:X::X:X> [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. 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.

3.17 STATIC

STATIC is a daemon that handles the installation and deletion of static routes.

3.17.1 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.

3.17.2 Static Route Commands

Static routing is a very fundamental feature of routing technology. It defines a static prefix and gateway.

ip route NETWORK GATEWAY table TABLENO nexthop-vrf VRFNAME DISTANCE vrf VRFNAME

ipv6 route NETWORK from SRCPREFIX GATEWAY table TABLENO nexthop-vrf VRFNAME DISTANCE vrf VI
NETWORK is destination prefix with a valid v4 or v6 network based upon initial form of the command. GATEWAY is gateway for the prefix it currently must match the v4 or v6 route type specified at the start of the command. GATEWAY can also be treated as an interface name. If the interface name is `null0` then zebra installs a blackhole route. 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. `nexthop-vrf` allows you to create a leaked route with a nexthop in the specified VRFNAME vrf VRFNAME allows you to create the route in a specified vrf. `nexthop-vrf` cannot be currently used with namespace based vrfs currently as well. The v6 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.

3.17.3 Multiple nexthop static route

To create multiple nexthops to the same NETWORK, 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 multihop 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 terminal
vrf r1-cust1
 ip route 10.0.0.0/24 10.0.0.2
exit-vrf
```

3.18 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.

3.18.1 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
```

no vnc nve-group NAME
Delete the NVE group named *name*.

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`

[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.

- 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 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
```

no vnc l2-group NAME
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

no 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 *bgp multiple-instance* and *router bgp AS view NAME* commands 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|os

vnc redistribute ipv4|ipv6 bgp-direct-to-nve-groups view VIEWNAME

no vnc redistribute ipv4|ipv6 bgp|bgp-direct|bgp-direct-to-nve-groups|connected|kernel|ospf
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

no 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.

3.18.2 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-
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
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
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|u
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.

3.18.3 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.

3.18.4 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.

`bgpd.conf` for `NVA 1` (192.168.1.100):

```
router bgp 64512

    bgp router-id 192.168.1.100

    neighbor 192.168.1.101  remote-as 64512
    neighbor 192.168.1.102  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
```

(continues on next page)

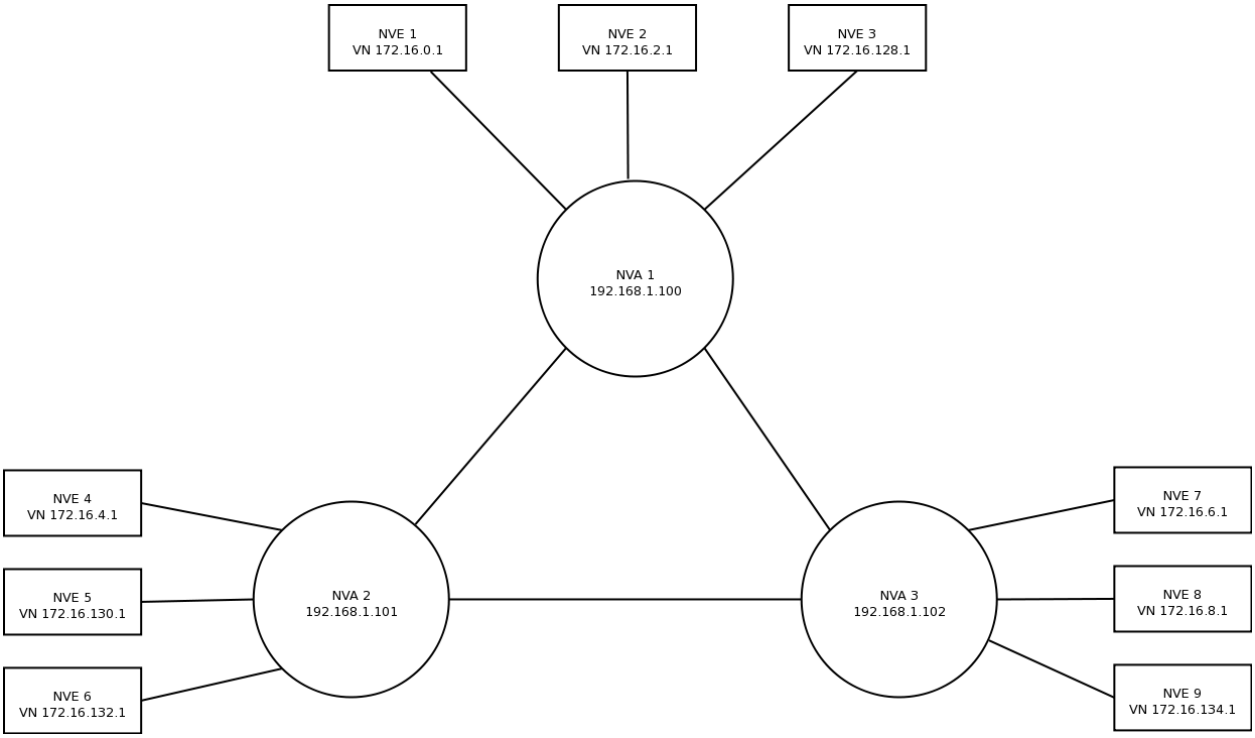


Fig. 5: A three-way full mesh with three NVEs per NVA.

(continued from previous page)

```
        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.

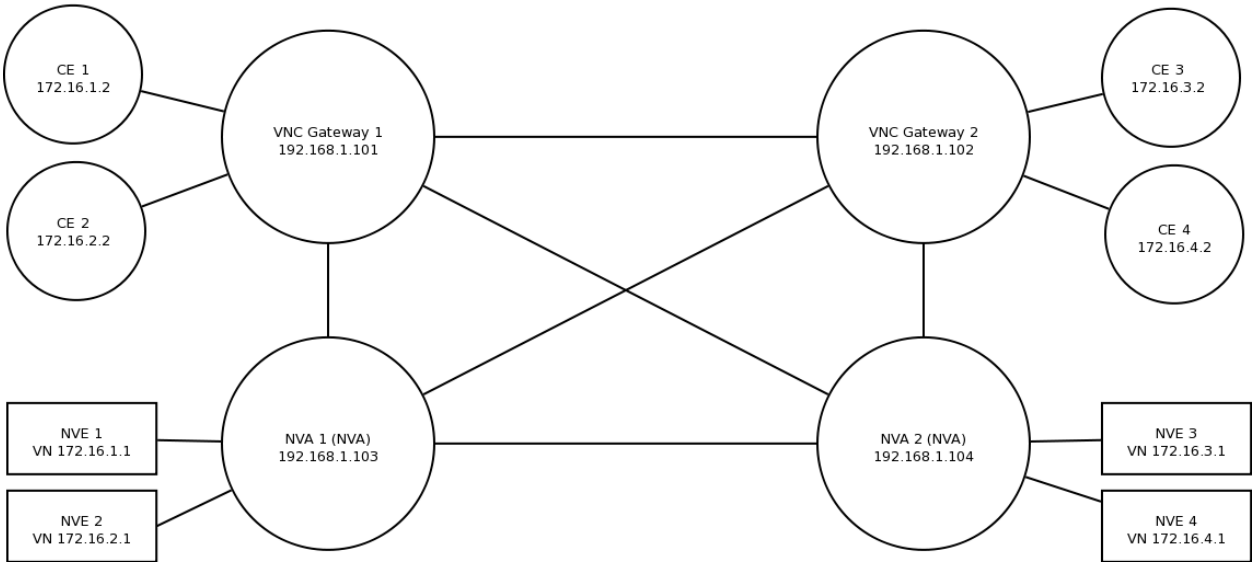


Fig. 6: Meshed NVEs and VNC-GWs

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
 exit-address-family
 !
 address-family ipv4 vpn
```

(continues on next page)

(continued from previous page)

```
  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).

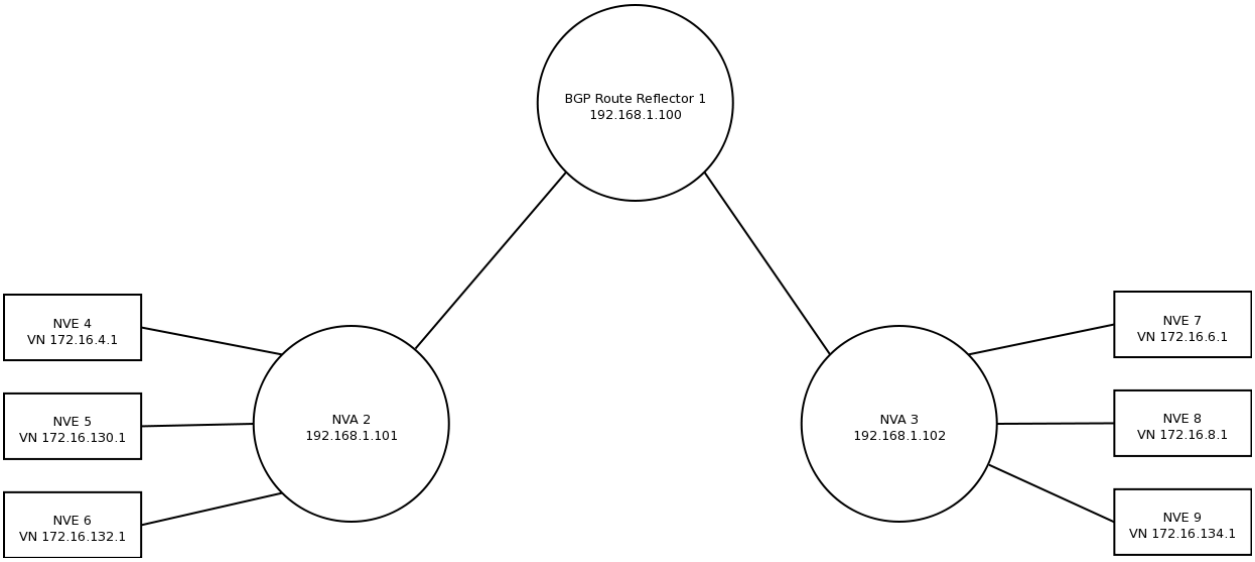


Fig. 7: Two NVAs and a BGP Route Reflector

`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
```

(continues on next page)

(continued from previous page)

```
        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.

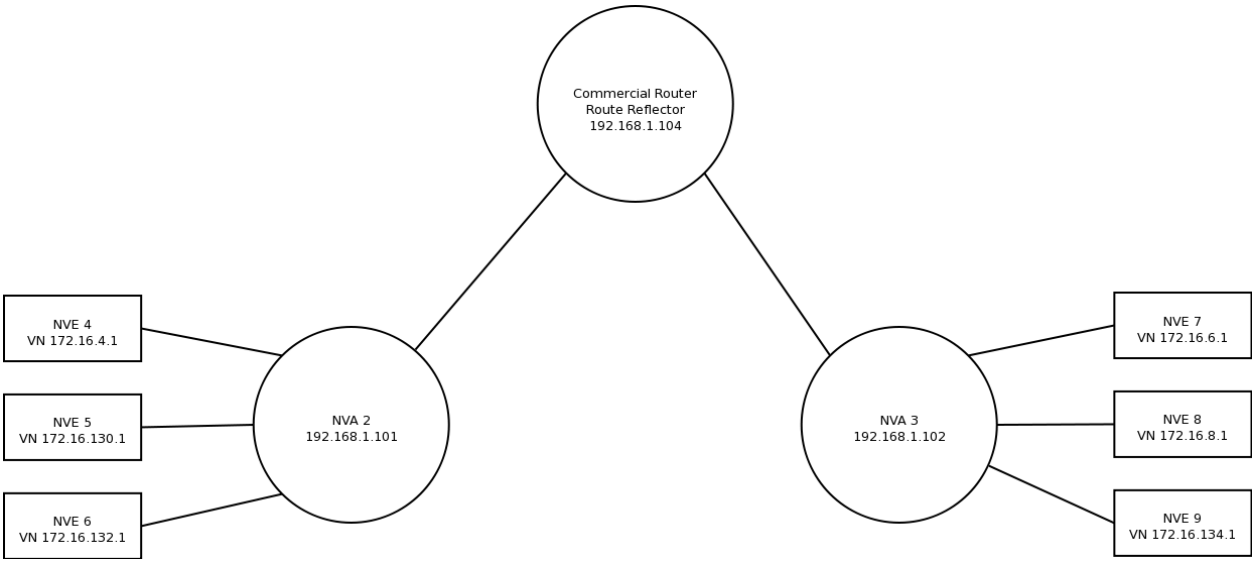


Fig. 8: Two NVAs with a commercial route reflector

`bgpd.conf` for BGP route reflector `Commercial Router` on 192.168.1.104::

```
version 8.5R1.13;
routing-options {
    rib inet.0 {
        static {
            route 172.16.0.0/16 next-hop 192.168.1.104;
        }
    }
    autonomous-system 64512;
    resolution {
        rib inet.3 {
```

(continues on next page)

(continued from previous page)

```
            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

    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
```

(continues on next page)

(continued from previous page)

```
    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

 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
```

(continues on next page)

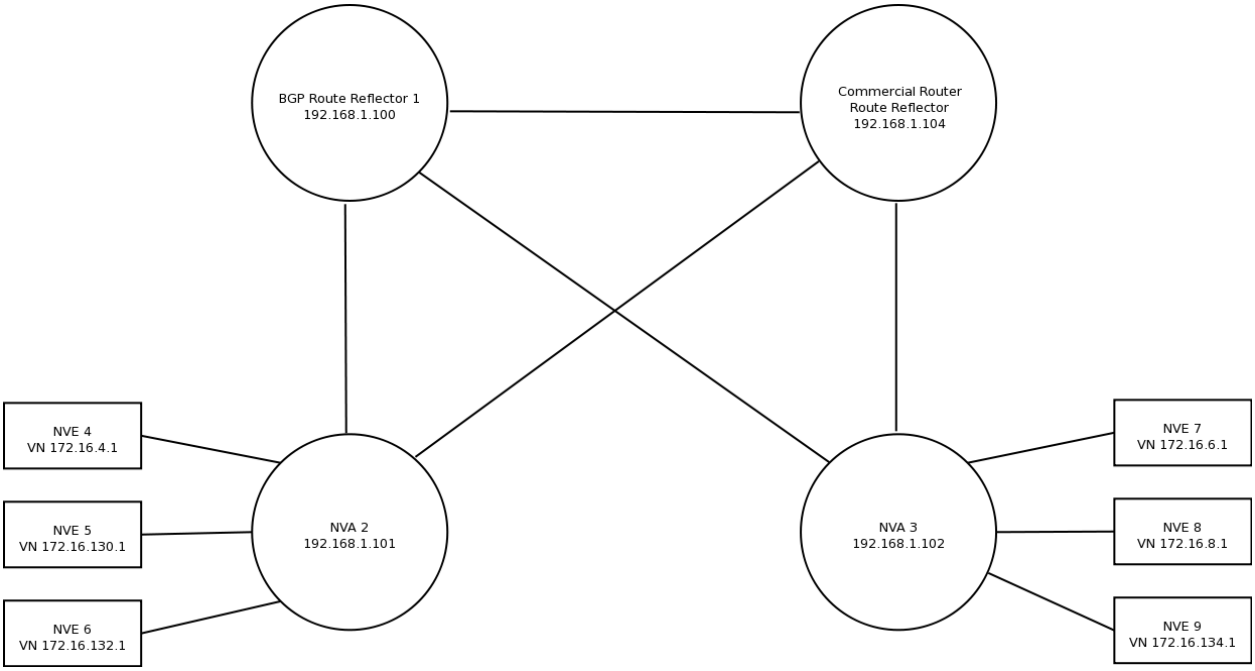


Fig. 9: FRR-based NVA with redundant route reflectors

(continued from previous page)

```
  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
 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
```

(continues on next page)

(continued from previous page)

```
 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;
        }
        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;
            }
```

(continues on next page)

(continued from previous page)

```
            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;
        }
    }
}
```

CHAPTER 4

Appendix

4.1 Reporting Bugs

This file describes the procedure for reporting FRRouting bugs. You are asked to follow this format when submitting bug reports.

Bugs submitted with woefully incomplete information will receive little attention and are likely to be closed. If you hit a suspected bug in an older version, you may be asked to test with a later version in your environment.

Often you may be asked for additional information to help solve the bug. Bugs may be closed after 30 days of non-response to requests to reconfirm or supply additional information.

Please report bugs on the project GitHub issue tracker at https://github.com/frrouting/frr/issues

4.1.1 Report Format & Requested Information

When reporting a bug, please provide the following information.

1. Your FRR version if it is a release build, or the commit hash if you built from source.
2. If you compiled from source, please provide your `./configure` line, including all option flags.
3. A full list of the FRR daemons you run.
4. Your platform name and version, e.g. `Ubuntu 18.04`.
5. Problem description.
   - Provide as much information as possible.
   - Copy and paste relevant commands and their output to describe your network setup.
   - Topology diagrams are helpful when reporting bugs involving more than one box.
   - Platform routing tables and interface configurations are useful if you are reporting a routing issue.

   Please be sure to review the provided information and censor any sensitive material.

6. All FRR configuration files you use. Again, please be sure to censor any sensitive information. For sensitive v4 / v6 addresses, we ask that you censor the inner octets; e.g., `192.XXX.XXX.32/24`.
7. If you are reporting a crash and have a core file, please supply a stack trace using GDB:

```
$ gdb exec_file core_file
  (gdb) bt .
```

8. Run all FRR daemons with full debugging on and send *only* the portion of logs which are relevant to your problem.
9. Patches, workarounds, and fixes are always welcome.

See also:

Basic Config Commands

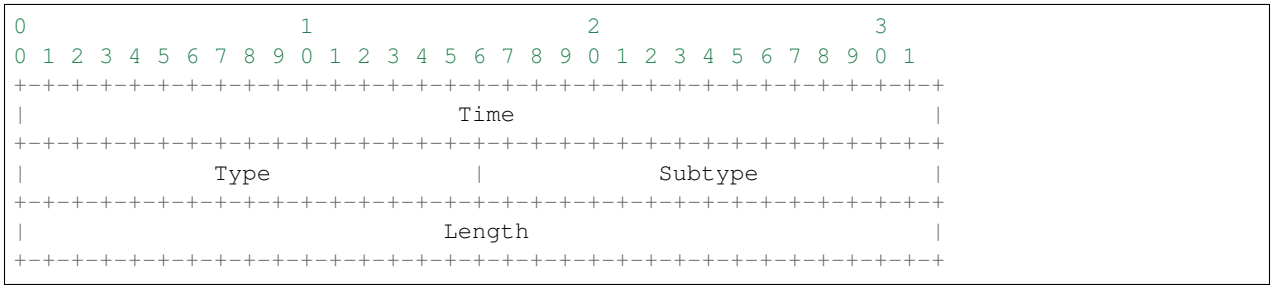
4.2 Packet Binary Dump Format

FRR can dump routing protocol packets into a file with a binary format.

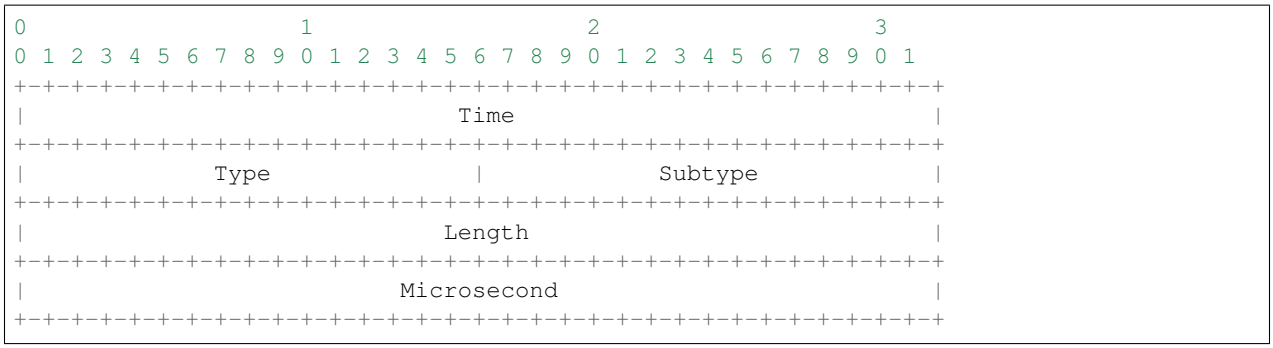
It seems to be better that we share the MRT's header format for backward compatibility with MRT's dump logs. We should also define the binary format excluding the header, because we must support both IP v4 and v6 addresses as socket addresses and / or routing entries.

In the last meeting, we discussed to have a version field in the header. But Masaki told us that we can define new 'type' value rather than having a 'version' field, and it seems to be better because we don't need to change header format.

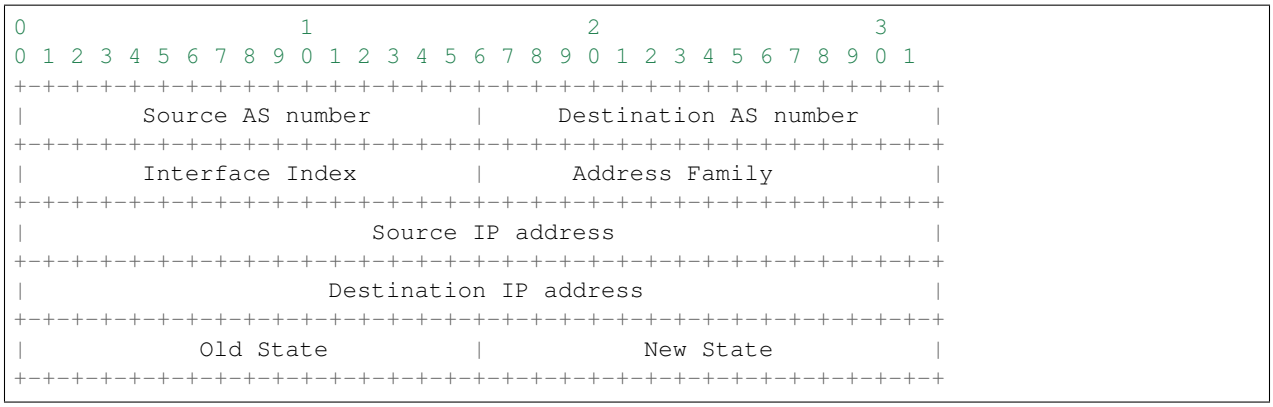
Here is the common header format. This is same as that of MRT.:



If 'type' is PROTOCOL_BGP4MP_ET, the common header format will contain an additional microsecond field (RFC6396 2011).:

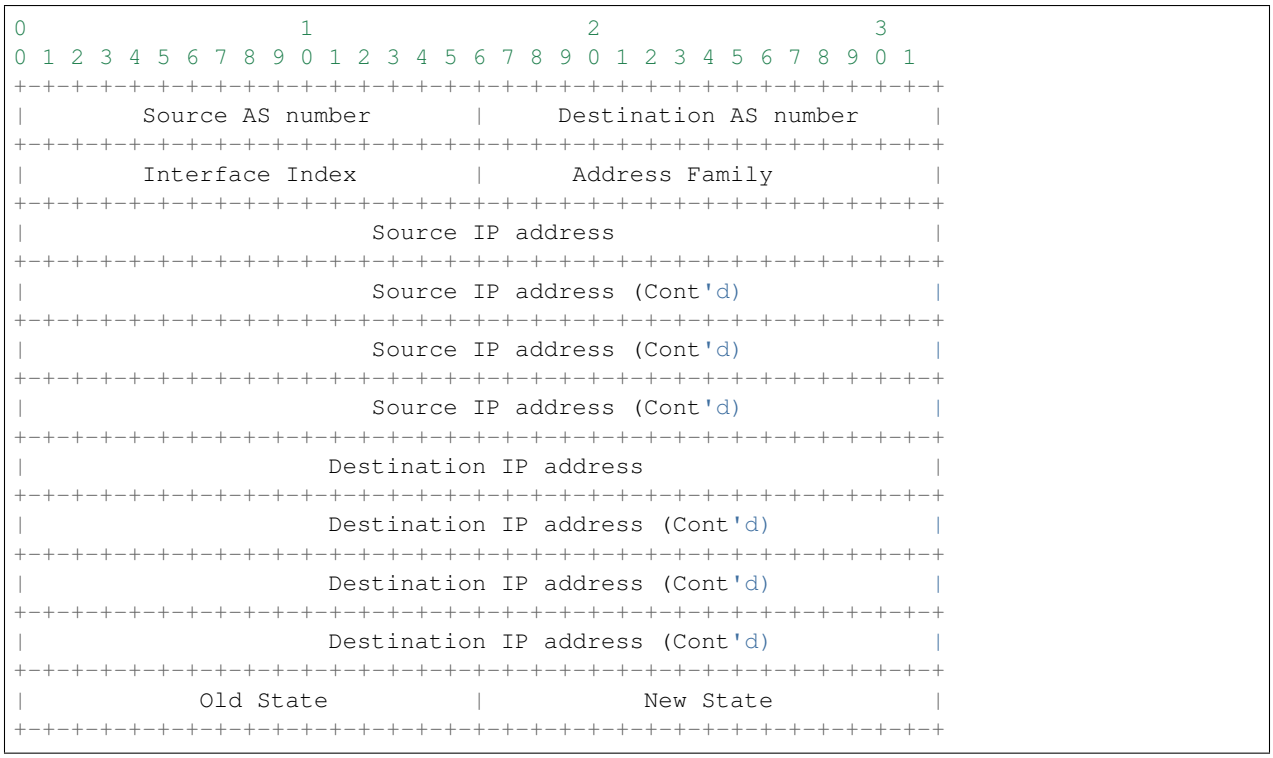


If 'type' is PROTOCOL_BGP4MP, 'subtype' is BGP4MP_STATE_CHANGE, and Address Family == IP (version 4):

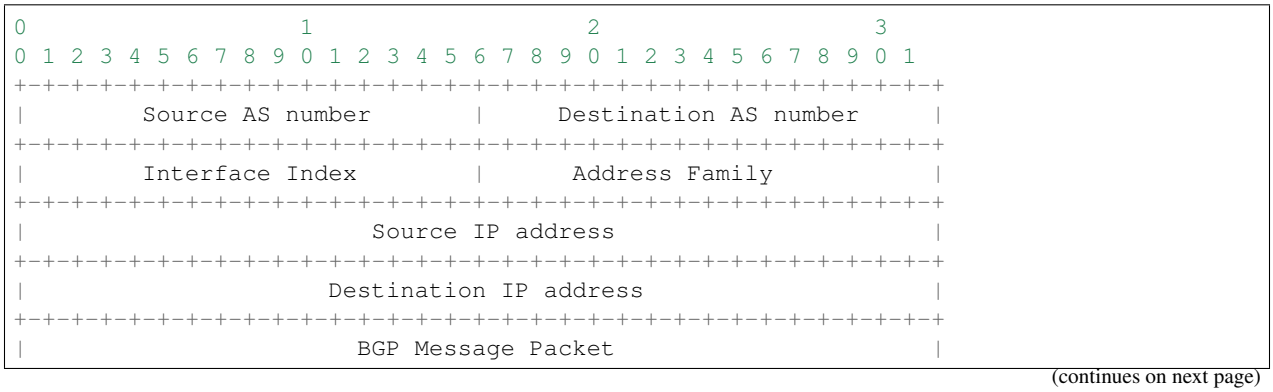


Where State is the value defined in RFC1771.

If 'type' is PROTOCOL_BGP4MP, 'subtype' is BGP4MP_STATE_CHANGE, and Address Family == IP version 6:



If 'type' is PROTOCOL_BGP4MP, 'subtype' is BGP4MP_MESSAGE, and Address Family == IP (version 4):



(continues on next page)

(continued from previous page)

```
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Where BGP Message Packet is the whole contents of the BGP4 message including header portion.

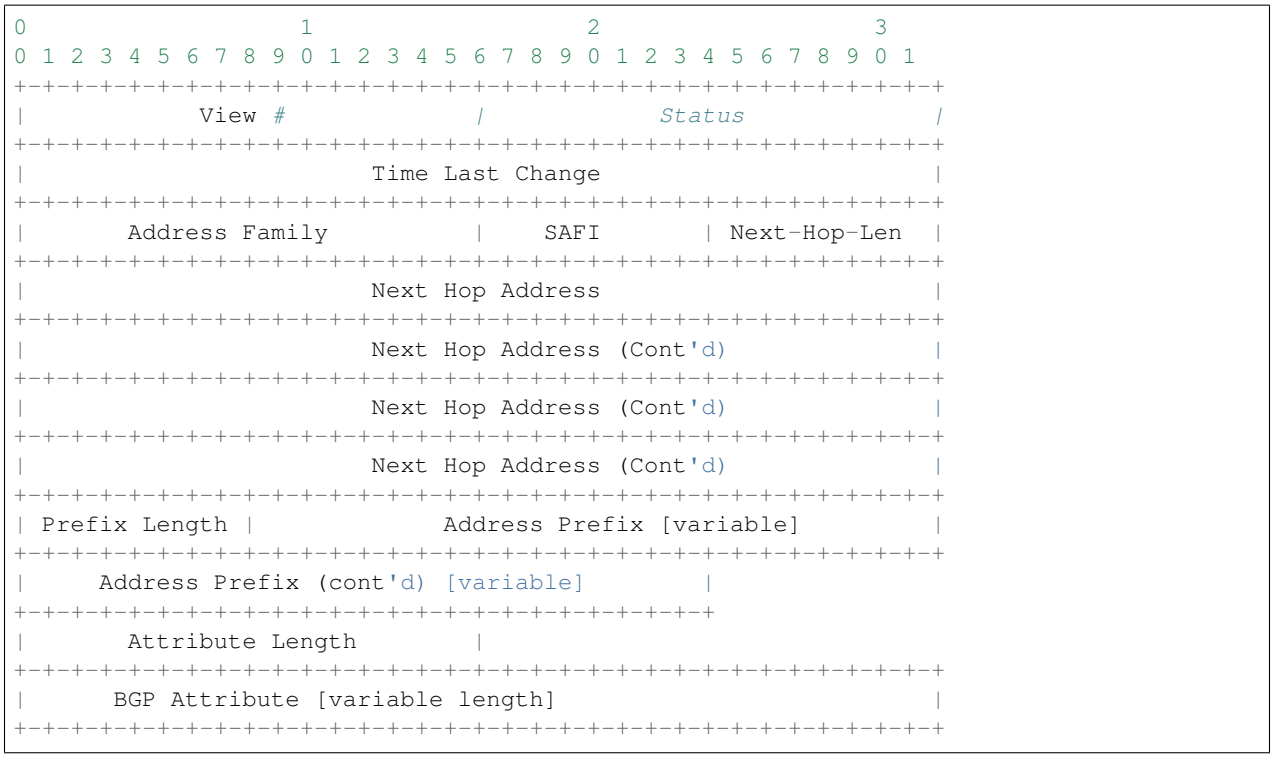
If 'type' is PROTOCOL_BGP4MP, 'subtype' is BGP4MP_MESSAGE, and Address Family == IP version 6:

```
0                   1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|        Source AS number       |     Destination AS number     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|        Interface Index        |      Address Family           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         Source IP address                     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         Source IP address (Cont'd)            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         Source IP address (Cont'd)            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         Source IP address (Cont'd)            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                      Destination IP address                   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                      Destination IP address (Cont'd)          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                      Destination IP address (Cont'd)          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                      Destination IP address (Cont'd)          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        BGP Message Packet                     |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

If 'type' is PROTOCOL_BGP4MP, 'subtype' is BGP4MP_ENTRY, and Address Family == IP (version 4):

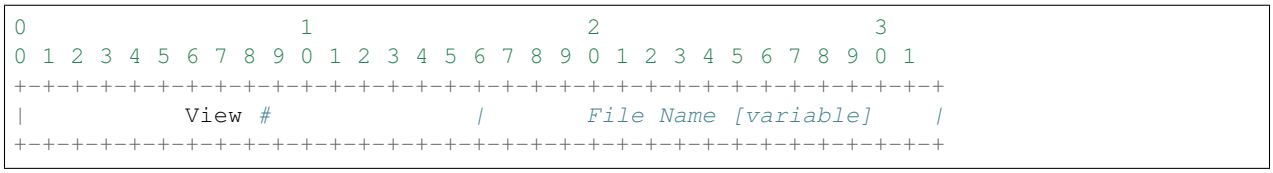
```
0                   1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|            View #            |            Status             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         Time Last Change                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|       Address Family          |    SAFI       | Next-Hop-Len  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         Next Hop Address                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Prefix Length |             Address Prefix [variable]         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|       Attribute Length        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|      BGP Attribute [variable length]                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

If 'type' is PROTOCOL_BGP4MP, 'subtype' is BGP4MP_ENTRY, and Address Family == IP version 6:



BGP4 Attribute must not contain MP_UNREACH_NLRI. If BGP Attribute has MP_REACH_NLRI field, it must has zero length NLRI, e.g., MP_REACH_NLRI has only Address Family, SAFI and next-hop values.

If 'type' is PROTOCOL_BGP4MP and 'subtype' is BGP4MP_SNAPSHOT:



The file specified in "File Name" contains all routing entries, which are in the format of `subtype == BGP4MP_ENTRY`.

```
Constants:

  /\* type value \*/
  #define MSG_PROTOCOL_BGP4MP    16
  #define MSG_PROTOCOL_BGP4MP_ET 17
  /\* subtype value \*/
  #define BGP4MP_STATE_CHANGE 0
  #define BGP4MP_MESSAGE 1
  #define BGP4MP_ENTRY 2
  #define BGP4MP_SNAPSHOT 3
```

4.3 Glossary

distance-vector A distance-vector routing protocol in data networks determines the best route for data packets based on distance. Distance-vector routing protocols measure the distance by the number of routers a packet has

to pass. Some distance-vector protocols also take into account network latency and other factors that influence traffic on a given route. To determine the best route across a network, routers on which a distance-vector protocol is implemented exchange information with one another, usually routing tables plus hop counts for destination networks and possibly other traffic information. Distance-vector routing protocols also require that a router informs its neighbours of network topology changes periodically. [distance-vector-rp]

link-state Link-state algorithms (also known as shortest path first algorithms) flood routing information to all nodes in the internetwork. Each router, however, sends only the portion of the routing table that describes the state of its own links. In link-state algorithms, each router builds a picture of the entire network in its routing tables. Distance vector algorithms (also known as Bellman-Ford algorithms) call for each router to send all or some portion of its routing table, but only to its neighbors. In essence, link-state algorithms send small updates everywhere, while distance vector algorithms send larger updates only to neighboring routers. Distance vector algorithms know only about their neighbors. [link-state-rp]

Bellman-Ford The Bellman–Ford algorithm is an algorithm that computes shortest paths from a single source vertex to all of the other vertices in a weighted digraph. [bellman-ford]

CHAPTER 5

Copyright notice

Bibliography

[Draft-IETF-uttaro-idr-bgp-persistence] <https://tools.ietf.org/id/draft-uttaro-idr-bgp-persistence-04.txt>

[Draft-IETF-agrewal-idr-accept-own-nexthop] <https://tools.ietf.org/id/draft-agrewal-idr-accept-own-nexthop-00.txt>

[Securing-BGP] Geoff Huston, Randy Bush: Securing BGP, In: The Internet Protocol Journal, Volume 14, No. 2, 2011. <http://www.cisco.com/web/about/ac123/ac147/archived_issues/ipj_14-2/142_bgp.html>

[Resource-Certification] Geoff Huston: Resource Certification, In: The Internet Protocol Journal, Volume 12, No.1, 2009. <http://www.cisco.com/web/about/ac123/ac147/archived_issues/ipj_12-1/121_resource.html>

[Draft-IETF-IDR-Flowspec-redirect-IP] <https://tools.ietf.org/id/draft-ietf-idr-flowspec-redirect-ip-02.txt>

[Draft-IETF-IDR-Flowspec-Interface-Set] <https://tools.ietf.org/id/draft-ietf-idr-flowspec-interfaceset-03.txt>

[Presentation] <https://docs.google.com/presentation/d/1ekQygUAG5yvQ3wWUyrw4Wcag0LgmbW1kV02IWcU4iUg/edit#slide=id.g378f0e1b5e_1_44>

[bgp-route-osci-cond] McPherson, D. and Gill, V. and Walton, D., "Border Gateway Protocol (BGP) Persistent Route Oscillation Condition", IETF RFC3345

[stable-flexible-ibgp] Flavel, A. and M. Roughan, "Stable and flexible iBGP", ACM SIGCOMM 2009

[ibgp-correctness] Griffin, T. and G. Wilfong, "On the correctness of IBGP configuration", ACM SIGCOMM 2002

[distance-vector-rp] https://en.wikipedia.org/wiki/Distance-vector_routing_protocol

[link-state-rp] https://en.wikipedia.org/wiki/Link-state_routing_protocol

[bellman-ford] https://en.wikipedia.org/wiki/Bellman-Ford_algorithm

Index

Symbols

Numbers

A

B

C

D

E

F

G

H

I

L

M

N

O

P

R

S

T

U

V

W

Z