Quagga

A routing software package for TCP/IP networks

Quagga 0.99.4

July 2006

Kunihiro Ishiguro, et al.

Copyright © 1999-2005 Kunihiro Ishiguro, et al.

Permission is granted to make and distribute verbatim copies of this manual provided the copyright notice and this permission notice are preserved on all copies.

Permission is granted to copy and distribute modified versions of this manual under the conditions for verbatim copying, provided that the entire resulting derived work is distributed under the terms of a permission notice identical to this one.

Permission is granted to copy and distribute translations of this manual into another language, under the above conditions for modified versions, except that this permission notice may be stated in a translation approved by Kunihiro Ishiguro.

Table of Contents

1 Overview

Quagga is a routing software package that provides TCP/IP based routing services with routing protocols support such as RIPv1, RIPv2, RIPng, OSPFv2, OSPFv3, BGP-4, and BGP-4+ (see Section 1.4 [Supported RFCs], page 3). Quagga also supports special BGP Route Reflector and Route Server behavior. In addition to traditional IPv4 routing protocols, Quagga also supports IPv6 routing protocols. With SNMP daemon which supports SMUX protocol, Quagga provides routing protocol MIBs (see Chapter 16 [SNMP Support], page 99).

Quagga uses an advanced software architecture to provide you with a high quality, multi server routing engine. Quagga 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 Quagga easily. You can use Quagga library as your program's client user interface.

Quagga is distributed under the GNU General Public License.

1.1 About Quagga

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 Quagga installed acts as a dedicated router. With Quagga, your machine exchanges routing information with other routers using routing protocols. Quagga 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 Quagga terminal interface.

Adding to routing protocol support, Quagga 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 Quagga 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 Quagga's dynamic routing protocol support for protocols such as RIP, OSPF 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. Quagga has a different system administration method. There are two user modes in Quagga. 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, Quagga supports common unicast routing protocols. Multicast routing protocols such as BGMP, PIM-SM, PIM-DM may be supported in Quagga 2.0. MPLS support is going on. In the future, TCP/IP filtering control, QoS control, diffserv configuration will be added to Quagga. Quagga project's final goal is making a productive, quality, free TCP/IP routing software.

1.2 System Architecture

Traditional routing software is made as a one process program which provides all of the routing protocol functionalities. Quagga takes a different approach. It is made from a collection of several daemons that work together to build the routing table. There may be several protocol-specific routing daemons and zebra the kernel routing manager.

The `ripd` daemon handles the RIP protocol, while `ospfd` is a daemon which supports OSPF version 2. `bgpd` supports the BGP-4 protocol. For changing the kernel routing table and for redistribution of routes between different routing protocols, there is a kernel routing table manager `zebra` daemon. It is easy to add a new routing protocol daemons to the entire routing system without affecting any other software. You need to run only the protocol daemon associated with routing protocols in use. Thus, user may run a specific daemon and send routing reports to a central routing console.

There is no need for these daemons to be running on the same machine. You can even run several same protocol daemons on the same machine. This architecture creates new possibilities for the routing system.

```
+----+  +----+  +-----+  +-----+
|bgpd|  |ripd|  |ospfd|  |zebra|
+----+  +----+  +-----+  +-----+
                            |
+---------------------------|--+
|                           v  |
|  UNIX Kernel  routing table  |
|                              |
+------------------------------+

    Quagga System Architecture
```

Multi-process architecture brings extensibility, modularity and maintainability. At the same time it also brings many configuration files and terminal interfaces. Each daemon has it's own configuration file and terminal interface. When you configure a static route, it must be done in `zebra` configuration file. When you configure BGP network it must be done in `bgpd` configuration file. This can be a very annoying thing. To resolve the problem, Quagga provides integrated user interface shell called `vtysh`. `vtysh` connects to each daemon with UNIX domain socket and then works as a proxy for user input.

Quagga was planned to use multi-threaded mechanism when it runs with a kernel that supports multi-threads. But at the moment, the thread library which comes with GNU/Linux or FreeBSD has some problems with running reliable services such as routing software, so we don't use threads at all. Instead we use the `select(2)` system call for multiplexing the events.

1.3 Supported Platforms

Currently Quagga supports GNU/Linux, BSD and Solaris. Porting Quagga to other platforms is not too difficult as platform dependent code should most be limited to the `zebra` daemon. Protocol daemons are mostly platform independent. Please let us know when you find out Quagga runs on a platform which is not listed below.

The list of officially supported platforms are listed below. Note that Quagga may run correctly on other platforms, and may run with partial functionality on further platforms.

- GNU/Linux 2.4.x and higher
- FreeBSD 4.x and higher
- NetBSD 1.6 and higher
- OpenBSD 2.5 and higher
- Solaris 8 and higher

1.4 Supported RFCs

Below is the list of currently supported RFC's.

RFC1058 *Routing Information Protocol. C.L. Hedrick. Jun-01-1988.*

RF2082 *RIP-2 MD5 Authentication. F. Baker, R. Atkinson. January 1997.*

RFC2453 *RIP Version 2. G. Malkin. November 1998.*

RFC2080 *RIPng for IPv6. G. Malkin, R. Minnear. January 1997.*

RFC2328 *OSPF Version 2. J. Moy. April 1998.*

RFC2370 *The OSPF Opaque LSA Option R. Coltun. July 1998.*

RFC3101 *The OSPF Not-So-Stubby Area (NSSA) Option P. Murphy. January 2003.*

RFC2740 *OSPF for IPv6. R. Coltun, D. Ferguson, J. Moy. December 1999.*

RFC1771 *A Border Gateway Protocol 4 (BGP-4). Y. Rekhter & T. Li. March 1995.*

RFC1965 *Autonomous System Confederations for BGP. P. Traina. June 1996.*

RFC1997 *BGP Communities Attribute. R. Chandra, P. Traina & T. Li. August 1996.*

RFC2545 *Use of BGP-4 Multiprotocol Extensions for IPv6 Inter-Domain Routing. P. Marques, F. Dupont. March 1999.*

RFC2796 *BGP Route Reflection An alternative to full mesh IBGP. T. Bates & R. Chandrasekeran. June 1996.*

RFC2858 *Multiprotocol Extensions for BGP-4. T. Bates, Y. Rekhter, R. Chandra, D. Katz. June 2000.*

RFC2842 *Capabilities Advertisement with BGP-4. R. Chandra, J. Scudder. May 2000.*

RFC3137 *OSPF Stub Router Advertisement, A. Retana, L. Nguyen, R. White, A. Zinin, D. McPherson. June 2001*

When SNMP support is enabled, below RFC is also supported.

RFC1227 *SNMP MUX protocol and MIB. M.T. Rose. May-01-1991.*

RFC1657 *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.*

RFC1724 *RIP Version 2 MIB Extension. G. Malkin & F. Baker. November 1994.*

RFC1850 *OSPF Version 2 Management Information Base. F. Baker, R. Coltun. November 1995.*

1.5 How to get Quagga

The official Quagga web-site is located at:

http://www.quagga.net/

and contains further information, as well as links to additional resources.

Quagga is a fork of GNU Zebra, whose web-site is located at:

http://www.zebra.org/.

1.6 Mailing List

There is a mailing list for discussions about Quagga. If you have any comments or suggestions to Quagga, please subscribe to:

http://lists.quagga.net/mailman/listinfo/quagga-users.

The Quagga site has further information on the available mailing lists, see:

http://www.quagga.net/lists.php

1.7 Bug Reports

If you think you have found a bug, please send a bug report to:

http://bugzilla.quagga.net

When you send a bug report, please be careful about the points below.

- Please note what kind of OS you are using. If you use the IPv6 stack please note that as well.
- Please show us the results of `netstat -rn` and `ifconfig -a`. Information from zebra's VTY command `show ip route` will also be helpful.
- Please send your configuration file with the report. If you specify arguments to the configure script please note that too.

Bug reports are very important for us to improve the quality of Quagga. Quagga is still in the development stage, but please don't hesitate to send a bug report to http://bugzilla.quagga.net.

2 Installation

There are three steps for installing the software: configuration, compilation, and installation.

The easiest way to get Quagga running is to issue the following commands:

```
% configure
% make
% make install
```

2.1 Configure the Software

2.1.1 The Configure script and its options

Quagga has an excellent configure script which automatically detects most host configurations. There are several additional configure options you can use to turn off IPv6 support, to disable the compilation of specific daemons, and to enable SNMP support.

'--enable-guile'
: Turn on compilation of the zebra-guile interpreter. You will need the guile library to make this. zebra-guile implementation is not yet finished. So this option is only useful for zebra-guile developers.

'--disable-ipv6'
: Turn off IPv6 related features and daemons. Quagga configure script automatically detects IPv6 stack. But sometimes you might want to disable IPv6 support of Quagga.

'--disable-zebra'
: Do not build zebra daemon.

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

'--enable-netlink'
: Force to enable GNU/Linux netlink interface. Quagga configure script detects netlink interface by checking a header file. When the header file does not match to the current running kernel, configure script will not turn on netlink support.

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

'--enable-opaque-lsa'
: Enable support for Opaque LSAs (RFC2370) in ospfd.

'--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.

'--enable-ospf-te'
: Enable support for OSPF Traffic Engineering Extension (internet-draft) this requires support for Opaque LSAs.

'--enable-multipath=*ARG*'
: Enable support for Equal Cost Multipath. *ARG* is the maximum number of ECMP paths to allow, set to 0 to allow unlimited number of paths.

'--enable-rtadv'
: Enable support IPV6 router advertisement in zebra.

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.

```
% ./configure --disable-ipv6
```

This command will configure zebra and the routing daemons.

2.1.2 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 Quagga daemons.

'--enable-user=*user*'
: Switch to user *ARG* shortly after startup, and run as user *ARG* in normal operation.

'--enable-group=*group*'
: Switch real and effective group to *group* shortly after startup.

'--enable-vty-group=*group*'
: Create Unix Vty sockets (for use with vtysh) with group owndership set to *group*. This allows one to create a seperate 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 'quagga' 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), the quagga system will retain only minimal capabilities required, further it will only raise these capabilities for brief periods. On systems without libcap, quagga will run as the user specified and only raise its uid back to uid 0 for brief periods.

2.1.3 Linux Notes

There are several options available only to GNU/Linux systems:[1]. If you use GNU/Linux, make sure that the current kernel configuration is what you want. Quagga will run with any kernel configuration but some recommendations do exist.

CONFIG_NETLINK
: Kernel/User netlink socket. This is a brand new feature which enables an advanced interface between the Linux kernel and zebra (see Chapter 15 [Kernel Interface], page 97).

CONFIG_RTNETLINK
: Routing messages. This makes it possible to receive netlink routing messages. If you specify this option, `zebra` can detect routing information updates directly from the kernel (see Chapter 15 [Kernel Interface], page 97).

CONFIG_IP_MULTICAST
: IP: multicasting. This option should be specified when you use `ripd` (see Chapter 5 [RIP], page 21) or `ospfd` (see Chapter 7 [OSPFv2], page 33) because these protocols use multicast.

IPv6 support has been added in GNU/Linux kernel version 2.2. If you try to use the Quagga IPv6 feature on a GNU/Linux kernel, please make sure the following libraries have been installed. Please note that these libraries will not be needed when you uses GNU C library 2.1 or upper.

`inet6-apps`
: The `inet6-apps` package includes basic IPv6 related libraries such as `inet_ntop` and `inet_pton`. Some basic IPv6 programs such as `ping`, `ftp`, and `inetd` are also included. The `inet-apps` can be found at `ftp://ftp.inner.net/pub/ipv6/`.

`net-tools`
: The `net-tools` package provides an IPv6 enabled interface and routing utility. It contains `ifconfig`, `route`, `netstat`, and other tools. `net-tools` may be found at `http://www.tazenda.demon.co.uk/phil/net-tools/`.

[1] GNU/Linux has very flexible kernel configuration features

2.2 Build the Software

After configuring the software, you will need to compile it for your system. Simply issue the command `make` in the root of the source directory and the software will be compiled. If you have *any* problems at this stage, be certain to send a bug report See Section 1.7 [Bug Reports], page 4.

```
% ./configure
.
.
.
./configure output
.
.
.
% make
```

2.3 Install the Software

Installing the software to your system consists of copying the compiled programs and supporting files to a standard location. After the installation process has completed, these files have been copied from your work directory to '`/usr/local/bin`', and '`/usr/local/etc`'.

To install the Quagga suite, issue the following command at your shell prompt: `make install`.

```
%
% make install
%
```

Quagga daemons have their own terminal interface or VTY. After installation, you have to setup each beast's port number to connect to them. Please 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
```

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/quagga/*.conf`'. See Section 3.1 [Config Commands], page 9.

3 Basic commands

There are five routing daemons in use, and there is one manager daemon. These daemons may be located on separate machines from the manager daemon. Each of these daemons will listen on a particular port for incoming VTY connections. The routing daemons are:

- `ripd`, `ripngd`, `ospfd`, `ospf6d`, `bgpd`
- `zebra`

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

3.1 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/quagga/*.conf`'

Each of the daemons has its own config file. For example, zebra's default config file name is:

'`/etc/quagga/zebra.conf`'

The daemon name plus '`.conf`' is the default config file name. You can specify a config file using the *-f* or *--config-file* options when starting the daemon.

3.1.1 Basic Config Commands

`hostname` *hostname* [Command]

Set hostname of the router.

`password` *password* [Command]

Set password for vty interface. If there is no password, a vty won't accept connections.

`enable password` *password* [Command]

Set enable password.

`log trap` *level* [Command]
`no log trap` [Command]

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

`log stdout` [Command]
`log stdout` *level* [Command]
`no log stdout` [Command]

Enable logging output to stdout. 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 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`.

`log file` *`filename`* [Command]
`log file` *`filename level`* [Command]
`no log file` [Command]

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

```
log file /var/log/quagga/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/quagga.<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.

`log syslog` [Command]
`log syslog` *`level`* [Command]
`no log syslog` [Command]

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.

`log monitor` [Command]
`log monitor` *`level`* [Command]
`no log monitor` [Command]

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, but can be changed using the deprecated `log trap` command) will be used. The `no` form of the command disables logging to terminal monitors.

`log facility` *`facility`* [Command]
`no log facility` [Command]

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

`log record-priority` [Command]
`no log record-priority` [Command]

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.

`service password-encryption` [Command]
Encrypt password.

`service advanced-vty` [Command]
Enable advanced mode VTY.

`service terminal-length` <*0-512*> [Command]
Set system wide line configuration. This configuration command applies to all VTY interfaces.

`line vty` [Command]
Enter vty configuration mode.

`banner motd default` [Command]
Set default motd string.

`no banner motd` [Command]
No motd banner string will be printed.

`exec-timeout` *minute* [Line Command]
`exec-timeout` *minute second* [Line Command]
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` [Line Command]
Do not perform timeout at all. This command is as same as `exec-timeout 0 0`.

`access-class` *access-list* [Line Command]
Restrict vty connections with an access list.

3.1.2 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'.

3.2 Terminal Mode Commands

`write terminal` [Command]
: Displays the current configuration to the vty interface.

`write file` [Command]
: Write current configuration to configuration file.

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

`terminal length` *<0-512>* [Command]
: Set terminal display length to *<0-512>*. If length is 0, no display control is performed.

`who` [Command]
: Show a list of currently connected vty sessions.

`list` [Command]
: List all available commands.

`show version` [Command]
: Show the current version of Quagga and its build host information.

`show logging` [Command]
: Shows the current configuration of the logging system. This includes the status of all logging destinations.

`logmsg` *level message* [Command]
: Send a message to all logging destinations that are enabled for messages of the given severity.

3.3 Common Invocation Options

These options apply to all Quagga daemons.

'`-d`'
'`--daemon`'
: Runs in daemon mode.

'`-f` *file*'
'`--config_file=`*file*'
: Set configuration file name.

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

'-i *file*'
'--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 Quagga to implement a routing looking glass. One machine can be used to collect differing routing views from differing points in the network.

'-A *address*'
'--vty_addr=*address*'
: Set the VTY local address to bind to. If set, the VTY socket will only be bound to this address.

'-P *port*'
'--vty_port=*port*'
: Set the VTY TCP port number. If set to 0 then the TCP VTY sockets will not be opened.

'-u *user*'
'--vty_addr=*user*'
: Set the user and group to run as.

'-v'
'--version'
: Print program version.

3.4 Virtual Terminal Interfaces

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

3.4.1 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 Quagga (version 0.99.4)
Copyright © 1999-2005 Kunihiro Ishiguro, et al.

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 running system information
  who               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#
```

'?' is very useful for looking up commands.

3.4.2 VTY Modes

There are three basic VTY modes:

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

3.4.2.1 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.

3.4.2.2 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.

3.4.2.3 VTY Other Modes

This page is for describing other modes.

3.4.3 VTY CLI Commands

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

3.4.3.1 CLI Movement Commands

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

C-f
⟨RIGHT⟩ Move forward one character.

C-b
⟨LEFT⟩ 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.

3.4.3.2 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.

3.4.3.3 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.

4 Zebra

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

4.1 Invoking zebra

Besides the common invocation options (see Section 3.3 [Common Invocation Options], page 12), 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, retain routes added by zebra.

4.2 Interface Commands

`interface` *`ifname`* [Command]

`shutdown` [Interface Command]
`no shutdown` [Interface Command]
Up or down the current interface.

`ip address` *`address/prefix`* [Interface Command]
`ip6 address` *`address/prefix`* [Interface Command]
`no ip address` *`address/prefix`* [Interface Command]
`no ip6 address` *`address/prefix`* [Interface Command]
Set the IPv4 or IPv6 address/prefix for the interface.

`ip address` *`address/prefix`* `secondary` [Interface Command]
`no ip address` *`address/prefix`* `secondary` [Interface Command]
Set the secondary flag for this address. This causes ospfd to not treat the address as a distinct subnet.

`description` *`description`* `...` [Interface Command]
Set description for the interface.

`multicast` [Interface Command]
`no multicast` [Interface Command]
Enable or disables multicast flag for the interface.

`bandwidth <1-10000000>` [Interface Command]
`no bandwidth <1-10000000>` [Interface Command]
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` [Interface Command]
`no link-detect` [Interface Command]

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.

4.3 Static Route Commands

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

`ip route` ***network gateway*** [Command]

network is destination prefix with format of A.B.C.D/M. *gateway* is gateway for the prefix. When *gateway* is A.B.C.D format. It is taken as a IPv4 address gateway. Otherwise it is treated as an interface name. If the interface name is *null0* then zebra installs a blackhole route.

```
ip route 10.0.0.0/8 10.0.0.2
ip route 10.0.0.0/8 ppp0
ip route 10.0.0.0/8 null0
```

First example defines 10.0.0.0/8 static route with gateway 10.0.0.2. Second one defines the same prefix but with gateway to interface ppp0. The third install a blackhole route.

`ip route` ***network netmask gateway*** [Command]

This is alternate version of above command. When *network* is A.B.C.D format, user must define *netmask* value with A.B.C.D format. *gateway* is same option as above command

```
ip route 10.0.0.0 255.255.255.0 10.0.0.2
ip route 10.0.0.0 255.255.255.0 ppp0
ip route 10.0.0.0 255.255.255.0 null0
```

These statements are equivalent to those in the previous example.

`ip route` ***network gateway distance*** [Command]

Installs the route with the specified distance.

Multiple nexthop static route

```
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.

```
zebra> 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-metric blackhole route, which can be useful to prevent traffic destined for a prefix to match less-specific routes (eg default) should the specified gateways not be reachable. Eg:

```
zebra> 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
```

`ipv6 route` *network gateway* [Command]
`ipv6 route` *network gateway distance* [Command]

These behave similarly to their ipv4 counterparts.

`table` *tableno* [Command]

Select the primary kernel routing table to be used. This only works for kernels supporting multiple routing tables (like GNU/Linux 2.2.x and later). After setting *tableno* with this command, static routes defined after this are added to the specified table.

4.4 zebra Terminal Mode Commands

`show ip route` [Command]

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` [Command]

`show interface` [Command]

`show ipforward` [Command]

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` [Command]

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

5 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 broadcasted to its neighboring router.

`ripd` supports RIP version 2 as described in RFC2453 and RIP version 1 as described in RFC1058.

5.1 Starting and Stopping ripd

The default configuration file name of `ripd`'s is '`ripd.conf`'. When invocation `ripd` searches directory /etc/quagga. 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 meaningss to `ripd`.

'`SIGHUP`' Reload configuration file '`ripd.conf`'. All configurations are reseted. All routes learned so far are cleared and removed from routing table.

'`SIGUSR1`' Rotate `ripd` logfile.

'`SIGINT`'
'`SIGTERM`' `ripd` sweeps all installed RIP routes then terminates properly.

`ripd` invocation options. Common options that can be specified (see Section 3.3 [Common Invocation Options], page 12).

'`-r`'
'`--retain`'
When the program terminates, retain routes added by `ripd`.

5.1.1 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. Quagga `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.

5.2 RIP Configuration

`router rip` [Command]

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` [Command]

Disable RIP.

`network` *network* [RIP Command]
`no network` *network* [RIP Command]

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* [RIP Command]
`no network` *ifname* [RIP Command]

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* [RIP Command]
`no neighbor` *a.b.c.d* [RIP Command]

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

`passive-interface (IFNAME|default)` [RIP command]
`no passive-interface IFNAME` [RIP command]

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.

RIP split-horizon

`ip split-horizon` [Interface command]
`no ip split-horizon` [Interface command]

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`.

5.3 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 overriden on a per-interface basis if needs be for send and receive seperately (see below).

It is important to note that RIPv1 can not 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 Section 5.9 [RIP Authentication], page 27.

`version version` [RIP Command]

Set RIP version to accept for reads and send. *version* can be either '1" or '2".

Disabling RIPv1 by specifying version 2 is STRONGLY encouraged, See Section 5.9 [RIP Authentication], page 27. This may become the default in a future release.

Default: Send Version 2, and accept either version.

`no version` [RIP Command]

Reset the global version setting back to the default.

`ip rip send version version` [Interface command]

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` [Interface command]

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).

5.4 How to Announce RIP route

`redistribute kernel` [RIP command]
`redistribute kernel metric <0-16>` [RIP command]
`redistribute kernel route-map` *route-map* [RIP command]
`no redistribute kernel` [RIP command]

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

`redistribute static` [RIP command]
`redistribute static metric <0-16>` [RIP command]
`redistribute static route-map` *route-map* [RIP command]
`no redistribute static` [RIP command]

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

`redistribute connected` [RIP command]
`redistribute connected metric <0-16>` [RIP command]
`redistribute connected route-map` *route-map* [RIP command]
`no redistribute connected` [RIP command]

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` [RIP command]
`redistribute ospf metric <0-16>` [RIP command]
`redistribute ospf route-map` *route-map* [RIP command]
`no redistribute ospf` [RIP command]

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

`redistribute bgp` [RIP command]
`redistribute bgp metric <0-16>` [RIP command]
`redistribute bgp route-map` *route-map* [RIP command]
`no redistribute bgp` [RIP command]

`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` [RIP command]

`route` *a.b.c.d/m* [RIP command]
`no route` *a.b.c.d/m* [RIP command]

This command is specific to Quagga. 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 Quagga and redistributing it in RIP using `redistribute static`.

5.5 Filtering RIP Routes

RIP routes can be filtered by a distribute-list.

`distribute-list` *access_list direct ifname* [Command]

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* [Command]

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.

5.6 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>` [RIP command]
`no default-metric <1-16>` [RIP command]

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)` [RIP command]
`offset-list` *access-list* `(in|out)` *ifname* [RIP command]

5.7 RIP distance

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

`distance <1-255>` [RIP command]
`no distance <1-255>` [RIP command]

Set default RIP distance to specified value.

`distance <1-255> A.B.C.D/M` [RIP command]
`no distance <1-255> A.B.C.D/M` [RIP command]

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` [RIP command]
`no distance <1-255> A.B.C.D/M access-list` [RIP command]

Set default RIP distance to specified value when the route's source IP address matches the specified prefix and the specified access-list.

5.8 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 Quagga'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 (see Chapter 13 [Route Map], page 89) is needed to use route-map functionality.

`match interface word` [Route Map]

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` [Route Map]
`match ip address prefix-list word` [Route Map]

Match if route destination is permitted by access-list.

`match ip next-hop A.B.C.D` [Route Map]

Cisco uses here <access-list>, `ripd` IPv4 address. Match if route has this next-hop (meaning next-hop listed in the rip route table - "show ip rip")

`match metric <0-4294967295>` [Route Map]

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` [Route Map]

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>` [Route Map]

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.

5.9 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, See Section 5.3 [RIP Version Control], page 23, 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, See Section 5.3 [RIP Version Control], page 23.

`ip rip authentication mode md5` [Interface command]
`no ip rip authentication mode md5` [Interface command]

Set the interface with RIPv2 MD5 authentication.

`ip rip authentication mode text` [Interface command]
`no ip rip authentication mode text` [Interface command]

Set the interface with RIPv2 simple password authentication.

`ip rip authentication string` *string* [Interface command]
`no ip rip authentication string` *string* [Interface command]

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* [Interface command]
`no ip rip authentication key-chain` *key-chain* [Interface command]

Specifiy Keyed MD5 chain.

```
!
key chain test
 key 1
  key-string test
!
```

```
interface eth1
 ip rip authentication mode md5
 ip rip authentication key-chain test
!
```

5.10 RIP Timers

`timers basic` *update timeout garbage* [RIP command]

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` [RIP command]

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

5.11 Show RIP Information

To display RIP routes.

`show ip rip` [Command]

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 protocols` [Command]

The command displays current RIP status. It includes RIP timer, filtering, version, RIP enabled interface and RIP peer inforation.

```
ripd> show ip protocols
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
```

5.12 RIP Debug Commands

Debug for RIP protocol.

`debug rip events` [Command]
  Debug rip events.

`debug rip` will show RIP events. Sending and receiving packets, timers, and changes in interfaces are events shown with `ripd`.

`debug rip packet` [Command]
  Debug rip packet.

`debug rip packet` will 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` [Command]
  Debug rip between zebra communication.

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` [Command]
  Display `ripd`'s debugging option.

`show debugging rip` will show all information currently set for ripd debug.

6 RIPng

`ripngd` supports the RIPng protocol as described in RFC2080. It's an IPv6 reincarnation of the RIP protocol.

6.1 Invoking ripngd

There are no `ripngd` specific invocation options. Common options can be specified (see Section 3.3 [Common Invocation Options], page 12).

6.2 ripngd Configuration

Currently ripngd supports the following commands:

`router ripng` [Command]
: Enable RIPng.

`flush_timer` *time* [RIPng Command]
: Set flush timer.

`network` *network* [RIPng Command]
: Set RIPng enabled interface by *network*

`network` *ifname* [RIPng Command]
: Set RIPng enabled interface by *ifname*

`route` *network* [RIPng Command]
: Set RIPng static routing announcement of *network*.

`router zebra` [Command]
: This command is the default and does not appear in the configuration. With this statement, RIPng routes go to the `zebra` daemon.

6.3 ripngd Terminal Mode Commands

`show ip ripng` [Command]

`show debugging ripng` [Command]

`debug ripng events` [Command]

`debug ripng packet` [Command]

`debug ripng zebra` [Command]

6.4 ripngd Filtering Commands

`distribute-list` *access_list* `(in|out)` *ifname* [Command]
: 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
```

7 OSPFv2

OSPF (Open Shortest Path First) version 2 is a routing protocol which is described in *RFC2328, OSPF Version 2*. OSPF is an IGP (Interior Gateway Protocol). Compared with RIP, OSPF can provide scalable network support and faster convergence times. OSPF is widely used in large networks such as ISP (Internet Service Provider) backbone and enterprise networks.

7.1 Configuring ospfd

There are no `ospfd` specific options. Common options can be specified (see Section 3.3 [Common Invocation Options], page 12) to `ospfd`. `ospfd` needs to 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`'.

7.2 OSPF router

To start OSPF process you have to specify the OSPF router. As of this writing, `ospfd` does not support multiple OSPF processes.

`router ospf` [Command]
`no router ospf` [Command]

Enable or disable the OSPF process. `ospfd` does not yet support multiple OSPF processes. So you can not specify an OSPF process number.

`ospf router-id` *a.b.c.d* [OSPF Command]
`no ospf router-id` [OSPF Command]

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* [OSPF Command]
`no ospf abr-type` *type* [OSPF Command]

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 Quagga, this restriction is lifted, allowing an ABR to consider summaries learnt 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 (see [OSPF virtual-link], page 37).

More information regarding the behaviour controlled by this command can be found in *RFC 3509, Alternative Implementations of OSPF Area Border Routers*, and *draft-ietf-ospf-shortcut-abr-02.txt*.

Quote: "Though the definition of the ABR (Area Border Router) 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` [OSPF Command]
`no ospf rfc1583compatibility` [OSPF Command]

RFC2328, the sucessor to *RFC1583*, 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 path are now of equal preference but still both preferred to external paths.

This command should NOT be set normally.

`log-adjacency-changes [detail]` [OSPF Command]
`no log-adjacency-changes [detail]` [OSPF Command]

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`* [OSPF Command]
`no passive-interface` *`interface`* [OSPF Command]

Do not speak OSPF interface on the given interface, but do advertise the interface as a stub link in the router-LSA (Link State Advertisement) 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 (see Section 7.5 [Redistribute routes to OSPF], page 40). This is the only way to advertise non-OSPF links into stub areas.

`timers throttle spf` *`delay initial-holdtime max-holdtime`* [OSPF Command]
`no timers throttle spf` [OSPF Command]

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 seperated 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 occuring then the current holdtime is re-set to the *initial-holdtime*. The current holdtime can be viewed with [show ip ospf], page 42, 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 seperated 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 supercedes the `timers spf` command in previous Quagga releases.

`max-metric router-lsa [on-startup|on-shutdown] <5-86400>` [OSPF Command]
`max-metric router-lsa administrative` [OSPF Command]
`no max-metric router-lsa [on-startup|on-shutdown|administrative]` [OSPF Command]

This enables *RFC3137, OSPF Stub Router Advertisement* 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], page 42 command.

`auto-cost reference-bandwidth <1-4294967>` [OSPF Command]
`no auto-cost reference-bandwidth` [OSPF Command]

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* [OSPF Command]
`network` *a.b.c.d/m* `area` *<0-4294967295>* [OSPF Command]
`no network` *a.b.c.d/m* `area` *a.b.c.d* [OSPF Command]
`no network` *a.b.c.d/m* `area` *<0-4294967295>* [OSPF Command]

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 (ie. 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.

7.3 OSPF area

`area` *a.b.c.d* `range` *a.b.c.d/m* [OSPF Command]
`area <0-4294967295> range` *a.b.c.d/m* [OSPF Command]
`no area` *a.b.c.d* `range` *a.b.c.d/m* [OSPF Command]
`no area <0-4294967295> range` *a.b.c.d/m* [OSPF Command]

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) (ie. 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 Quagga.

```
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 (ie. described with router or network LSA) from this range.

`area` *a.b.c.d* `range IPV4_PREFIX not-advertise` [OSPF Command]
`no area` *a.b.c.d* `range IPV4_PREFIX not-advertise` [OSPF Command]

Instead of summarizing intra area paths filter them - ie. 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` [OSPF Command]
`no area` *a.b.c.d* `range IPV4_PREFIX substitute IPV4_PREFIX` [OSPF Command]

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 (ie. 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* [OSPF Command]
`area <0-4294967295> virtual-link` *a.b.c.d* [OSPF Command]
`no area` *a.b.c.d* `virtual-link` *a.b.c.d* [OSPF Command]
`no area <0-4294967295> virtual-link` *a.b.c.d* [OSPF Command]

`area` *a.b.c.d* `shortcut` [OSPF Command]
`area <0-4294967295> shortcut` [OSPF Command]
`no area` *a.b.c.d* `shortcut` [OSPF Command]
`no area <0-4294967295> shortcut` [OSPF Command]

Configure the area as Shortcut capable. See *RFC3509*. This requires that the 'abr-type' be set to 'shortcut'.

`area` *a.b.c.d* `stub` [OSPF Command]
`area <0-4294967295> stub` [OSPF Command]
`no area` *a.b.c.d* `stub` [OSPF Command]
`no area <0-4294967295> stub` [OSPF Command]

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` [OSPF Command]
`area <0-4294967295> stub no-summary` [OSPF Command]
`no area` *a.b.c.d* `stub no-summary` [OSPF Command]
`no area <0-4294967295> stub no-summary` [OSPF Command]

Prevents an `ospfd` ABR from injecting inter-area summaries into the specified stub area.

`area` *a.b.c.d* `default-cost <0-16777215>` [OSPF Command]
`no area` *a.b.c.d* `default-cost <0-16777215>` [OSPF Command]

Set the cost of default-summary LSAs announced to stubby areas.

`area` *a.b.c.d* `export-list NAME` [OSPF Command]
`area <0-4294967295> export-list NAME` [OSPF Command]
`no area` *a.b.c.d* `export-list NAME` [OSPF Command]
`no area <0-4294967295> export-list NAME` [OSPF Command]

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` [OSPF Command]
`area <0-4294967295> import-list NAME` [OSPF Command]
`no area` *a.b.c.d* `import-list NAME` [OSPF Command]
`no area <0-4294967295> import-list NAME` [OSPF Command]

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` [OSPF Command]
`area` *a.b.c.d* `filter-list prefix NAME out` [OSPF Command]
`area <0-4294967295> filter-list prefix NAME in` [OSPF Command]
`area <0-4294967295> filter-list prefix NAME out` [OSPF Command]
`no area` *a.b.c.d* `filter-list prefix NAME in` [OSPF Command]
`no area` *a.b.c.d* `filter-list prefix NAME out` [OSPF Command]
`no area <0-4294967295> filter-list prefix NAME in` [OSPF Command]
`no area <0-4294967295> filter-list prefix NAME out` [OSPF Command]

Filtering Type-3 summary-LSAs to/from area using prefix lists. This command makes sense in ABR only.

`area` *a.b.c.d* `authentication` [OSPF Command]
`area <0-4294967295> authentication` [OSPF Command]
`no area` *a.b.c.d* `authentication` [OSPF Command]
`no area <0-4294967295> authentication` [OSPF Command]

Specify that simple password authentication should be used for the given area.

`area` *a.b.c.d* `authentication message-digest` [OSPF Command]
`area <0-4294967295> authentication message-digest` [OSPF Command]

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 (see [ip ospf message-digest-key], page 39).

MD5 authentication may also be configured on a per-interface basis (see [ip ospf authentication message-digest], page 39). Such per-interface settings will override any per-area authentication setting.

7.4 OSPF interface

`ip ospf authentication-key AUTH_KEY` [Interface Command]
`no ip ospf authentication-key` [Interface Command]

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 (see [ip ospf authentication message-digest], page 39).

`ip ospf authentication message-digest` [Interface Command]

Specify that MD5 HMAC authentication must be used on this interface. MD5 keying material must also be configured (see [ip ospf message-digest-key], page 39). Overrides any authentication enabled on a per-area basis (see [area authentication message-digest], page 38).

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 reestabish adjacencies with its neighbours after restarts/reboots. The host should have system time be set at boot from an external or non-volatile source (eg battery backed clock, NTP, etc.) or else the system clock should be periodically saved to non-volative storage and restored at boot if MD5 authentication is to be expected to work reliably.

`ip ospf message-digest-key KEYID md5 KEY` [Interface Command]
`no ip ospf message-digest-key` [Interface Command]

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>` [Interface Command]
`no ip ospf cost` [Interface Command]

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>` [Interface Command]
`ip ospf dead-interval minimal hello-multiplier <2-20>` [Interface Command]
`no ip ospf dead-interval` [Interface Command]

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>` [Interface Command]
`no ip ospf hello-interval` [Interface Command]

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], page 39 is also specified for the interface.

`ip ospf network (broadcast|non-broadcast|point-to-multipoint|point-to-point)` [Interface Command]
`no ip ospf network` [Interface Command]

Set explicitly network type for specifed interface.

`ip ospf priority <0-255>` [Interface Command]
`no ip ospf priority` [Interface Command]

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>` [Interface Command]
`no ip ospf retransmit interval` [Interface Command]

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` [Interface Command]
`no ip ospf transmit-delay` [Interface Command]

Set number of seconds for InfTransDelay value. LSAs' age should be incremented by this value when transmitting. The default value is 1 seconds.

7.5 Redistribute routes to OSPF

`redistribute (kernel|connected|static|rip|bgp)` [OSPF Command]
`redistribute (kernel|connected|static|rip|bgp) route-map` [OSPF Command]
`redistribute (kernel|connected|static|rip|bgp) metric-type (1|2)` [OSPF Command]
`redistribute (kernel|connected|static|rip|bgp) metric-type (1|2) route-map word` [OSPF Command]
`redistribute (kernel|connected|static|rip|bgp) metric <0-16777214>` [OSPF Command]
`redistribute (kernel|connected|static|rip|bgp) metric <0-16777214> route-map word` [OSPF Command]
`redistribute (kernel|connected|static|rip|bgp) metric-type (1|2) metric <0-16777214>` [OSPF Command]

`redistribute (kernel|connected|static|rip|bgp) metric-type (1|2) metric <0-16777214> route-map` *word* [OSPF Command]

`no redistribute (kernel|connected|static|rip|bgp)` [OSPF Command]

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], page 41.

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 *passive-interface*, see [OSPF passive-interface], page 34.

`default-information originate` [OSPF Command]

`default-information originate metric <0-16777214>` [OSPF Command]

`default-information originate metric <0-16777214> metric-type (1|2)` [OSPF Command]

`default-information originate metric <0-16777214> metric-type (1|2) route-map` *word* [OSPF Command]

`default-information originate always` [OSPF Command]

`default-information originate always metric <0-16777214>` [OSPF Command]

`default-information originate always metric <0-16777214> metric-type (1|2)` [OSPF Command]

`default-information originate always metric <0-16777214> metric-type (1|2) route-map` *word* [OSPF Command]

`no default-information originate` [OSPF Command]

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` [OSPF Command]

`no distribute-list NAME out (kernel|connected|static|rip|ospf` [OSPF Command]

Apply the access-list filter, NAME, to redistributed routes of the given type before allowing the routes to redistributed into OSPF (see [OSPF redistribute], page 41).

`default-metric <0-16777214>` [OSPF Command]

`no default-metric` [OSPF Command]

`distance <1-255>` [OSPF Command]

`no distance <1-255>` [OSPF Command]

`distance ospf (intra-area|inter-area|external) <1-255>` [OSPF Command]

`no distance ospf` [OSPF Command]

`router zebra` [Command]

`no router zebra` [Command]

7.6 Showing OSPF information

`show ip ospf` [Command]

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

`show ip ospf interface [INTERFACE]` [Command]

Show state and configuration of OSPF the specified interface, or all interfaces if no interface is given.

`show ip ospf neighbor` [Command]
`show ip ospf neighbor INTERFACE` [Command]
`show ip ospf neighbor detail` [Command]
`show ip ospf neighbor INTERFACE detail` [Command]

`show ip ospf database` [Command]

`show ip ospf database (asbr-summary|external|network|router|summary)` [Command]
`show ip ospf database (asbr-summary|external|network|router|summary)` *link-state-id* [Command]
`show ip ospf database (asbr-summary|external|network|router|summary)` *link-state-id* `adv-router` *adv-router* [Command]
`show ip ospf database (asbr-summary|external|network|router|summary) adv-router` *adv-router* [Command]
`show ip ospf database (asbr-summary|external|network|router|summary)` *link-state-id* `self-originate` [Command]
`show ip ospf database (asbr-summary|external|network|router|summary) self-originate` [Command]

`show ip ospf database max-age` [Command]

`show ip ospf database self-originate` [Command]

`show ip ospf route` [Command]

Show the OSPF routing table, as determined by the most recent SPF calculation.

7.7 Debugging OSPF

`debug ospf packet (hello|dd|ls-request|ls-update|ls-ack|all) (send|recv) [detail]` [Command] no debug ospf packet (hello|dd|ls-request|ls-update|ls-ack|all) (send|recv) [detail] [Command]

debug ospf ism [Command]
debug ospf ism (status|events|timers) [Command]
no debug ospf ism [Command]
no debug ospf ism (status|events|timers) [Command]

debug ospf nsm [Command]
debug ospf nsm (status|events|timers) [Command]
no debug ospf nsm [Command]
no debug ospf nsm (status|events|timers) [Command]

debug ospf lsa [Command]
debug ospf lsa (generate|flooding|refresh) [Command]
no debug ospf lsa [Command]
no debug ospf lsa (generate|flooding|refresh) [Command]

debug ospf zebra [Command]
debug ospf zebra (interface|redistribute) [Command]
no debug ospf zebra [Command]
no debug ospf zebra (interface|redistribute) [Command]

show debugging ospf [Command]

7.8 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/quagga/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
 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
!
```

8 OSPFv3

`ospf6d` is a daemon support OSPF version 3 for IPv6 network. OSPF for IPv6 is described in RFC2740.

8.1 OSPF6 router

`router ospf6` [Command]

`router-id` *a.b.c.d* [OSPF6 Command]

Set router's Router-ID.

`interface` *ifname* `area` *area* [OSPF6 Command]

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

8.2 OSPF6 area

Area support for OSPFv3 is not yet implemented.

8.3 OSPF6 interface

`ipv6 ospf6 cost COST` [Interface Command]

Sets interface's output cost. Default value is 1.

`ipv6 ospf6 hello-interval HELLOINTERVAL` [Interface Command]

Sets interface's Hello Interval. Default 40

`ipv6 ospf6 dead-interval DEADINTERVAL` [Interface Command]

Sets interface's Router Dead Interval. Default value is 40.

`ipv6 ospf6 retransmit-interval RETRANSMITINTERVAL` [Interface Command]

Sets interface's Rxmt Interval. Default value is 5.

`ipv6 ospf6 priority PRIORITY` [Interface Command]

Sets interface's Router Priority. Default value is 1.

`ipv6 ospf6 transmit-delay TRANSMITDELAY` [Interface Command]

Sets interface's Inf-Trans-Delay. Default value is 1.

8.4 Redistribute routes to OSPF6

`redistribute static` [OSPF6 Command]
`redistribute connected` [OSPF6 Command]
`redistribute ripng` [OSPF6 Command]

8.5 Showing OSPF6 information

`show ipv6 ospf6 [INSTANCE_ID]` [Command]

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` [Command]

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

`show ipv6 ospf6 interface` [Command]

To see OSPF interface configuration like costs.

`show ipv6 ospf6 neighbor` [Command]

Shows state and chosen (Backup) DR of neighbor.

`show ipv6 ospf6 request-list A.B.C.D` [Command]

Shows requestlist of neighbor.

`show ipv6 route ospf6` [Command]

This command shows internal routing table.

8.6 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
!
```

9 BGP

BGP stands for a Border Gateway Protocol. The lastest BGP version is 4. It is referred as BGP-4. BGP-4 is one of the Exterior Gateway Protocols and de-fact standard of Inter Domain routing protocol. BGP-4 is described in *RFC1771, A Border Gateway Protocol 4 (BGP-4)*.

Many extensions have been added to *RFC1771*. *RFC2858, Multiprotocol Extensions for BGP-4* provides multiprotocol support to BGP-4.

9.1 Starting BGP

Default configuration file of `bgpd` is '`bgpd.conf`'. `bgpd` searches the current directory first then /etc/quagga/bgpd.conf. All of bgpd's command must be configured in '`bgpd.conf`'.

`bgpd` specific invocation options are described below. Common options may also be specified (see Section 3.3 [Common Invocation Options], page 12).

'`-p` *PORT*'
'`--bgp_port=`*PORT*'
Set the bgp protocol's port number.

'`-r`'
'`--retain`'
When program terminates, retain BGP routes added by zebra.

9.2 BGP router

First of all you must configure BGP router with `router bgp` command. To configure BGP router, you need AS number. AS number is an identification of autonomous system. BGP protocol uses the AS number for detecting whether the BGP connection is internal one or external one.

`router bgp` *asn* [Command]
Enable a BGP protocol process with the specified *asn*. After this statement you can input any `BGP Commands`. You can not create different BGP process under different *asn* without specifying `multiple-instance` (see Section 9.13.1 [Multiple instance], page 61).

`no router bgp` *asn* [Command]
Destroy a BGP protocol process with the specified *asn*.

`bgp router-id` *A.B.C.D* [BGP]
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.

9.2.1 BGP distance

`distance bgp <1-255> <1-255> <1-255>` [BGP]
This command change distance value of BGP. Each argument is distance value for external routes, internal routes and local routes.

`distance <1-255> A.B.C.D/M` [BGP]
`distance <1-255> A.B.C.D/M word` [BGP]
This command set distance value to

9.2.2 BGP decision process

1. Weight check
2. Local preference check.
3. Local route check.
4. AS path length check.
5. Origin check.
6. MED check.

`bgp bestpath as-path confed` [BGP]
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.

9.3 BGP network

9.3.1 BGP route

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

```
router bgp 1
 network 10.0.0.0/8
```

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; `bgp` doesn't care about IGP routes when announcing its routes.

`no network A.B.C.D/M` [BGP]

9.3.2 Route Aggregation

`aggregate-address A.B.C.D/M` [BGP]
This command specifies an aggregate address.

`aggregate-address A.B.C.D/M as-set` [BGP]
This command specifies an aggregate address. Resulting routes inlucde AS set.

`aggregate-address A.B.C.D/M summary-only` [BGP]
This command specifies an aggregate address. Aggreated routes will not be announce.

`no aggregate-address A.B.C.D/M` [BGP]

9.3.3 Redistribute to BGP

`redistribute kernel` [BGP]
Redistribute kernel route to BGP process.

`redistribute static` [BGP]
Redistribute static route to BGP process.

`redistribute connected` [BGP]

Redistribute connected route to BGP process.

`redistribute rip` [BGP]

Redistribute RIP route to BGP process.

`redistribute ospf` [BGP]

Redistribute OSPF route to BGP process.

9.4 BGP Peer

9.4.1 Defining Peer

`neighbor` *`peer`* `remote-as` *`asn`* [BGP]

Creates a new neighbor whose remote-as is *asn*. *peer* can be an IPv4 address or an IPv6 address.

```
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
```

9.4.2 BGP Peer commands

In a `router bgp` clause there are neighbor specific configurations required.

`neighbor` *`peer`* `shutdown` [BGP]
`no neighbor` *`peer`* `shutdown` [BGP]

Shutdown the peer. We can delete the neighbor's configuration by `no neighbor` *`peer`* `remote-as` *`as-number`* 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.

`neighbor` *`peer`* `ebgp-multihop` [BGP]
`no neighbor` *`peer`* `ebgp-multihop` [BGP]

`neighbor` *`peer`* `description ...` [BGP]
`no neighbor` *`peer`* `description ...` [BGP]

Set description of the peer.

`neighbor` *`peer`* `version` *`version`* [BGP]

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.

`neighbor peer interface ifname` [BGP]
`no neighbor peer interface ifname` [BGP]

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.

`neighbor peer next-hop-self` [BGP]
`no neighbor peer next-hop-self` [BGP]

This command specifies an announced route's nexthop as being equivalent to the address of the bgp router.

`neighbor peer update-source <ifname|address>` [BGP]
`no neighbor peer update-source` [BGP]

Specify the IPv4 source address to use for the BGP session to this neighbour, may be specified as either an IPv4 address directly or as an interface name (in which case the `zebra` daemon MUST be running in order for `bgpd` to be able to retrieve interface state).

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

`neighbor peer default-originate` [BGP]
`no neighbor peer default-originate` [BGP]

`bgpd`'s default is to not announce the default route (0.0.0.0/0) even it is in routing table. When you want to announce default routes to the peer, use this command.

`neighbor peer port port` [BGP]
`neighbor peer port port` [BGP]

`neighbor peer send-community` [BGP]
`neighbor peer send-community` [BGP]

`neighbor peer weight weight` [BGP]
`no neighbor peer weight weight` [BGP]

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

`neighbor peer maximum-prefix number` [BGP]
`no neighbor peer maximum-prefix number` [BGP]

9.4.3 Peer filtering

`neighbor peer distribute-list name [in|out]` [BGP]

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

`neighbor peer prefix-list name [in|out]` [BGP command]

`neighbor peer filter-list name [in|out]` [BGP command]

`neighbor peer route-map name [in|out]` [BGP]

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

9.5 BGP Peer Group

`neighbor word peer-group` [BGP]

This command defines a new peer group.

`neighbor peer peer-group word` [BGP]

This command bind specific peer to peer group *word*.

9.6 BGP Address Family

9.7 Autonomous System

The AS (Autonomous System) number is one of the essential element of BGP. BGP is a distance vector routing protocol, and the AS-Path framework provides distance vector metric and loop detection to BGP. *RFC1930, Guidelines for creation, selection, and registration of an Autonomous System (AS)* provides some background on the concepts of an AS.

The AS number 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 to be advertised in the global Internet.

9.7.1 AS Path Regular Expression

AS path regular expression can be used for displaying BGP routes and AS path access list. AS path regular expression is based on `POSIX 1003.2` regular expressions. Following description is just a subset of `POSIX` regular expression. User can use full `POSIX` regular expression. Adding to that special character '_' is added for AS path regular expression.

`.` 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.

`_` Character `_` has special meanings in AS path regular expression. 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. `show ip bgp regexp _7675_` matches to all of BGP routes which as AS number include *7675*.

9.7.2 Display BGP Routes by AS Path

To show BGP routes which has specific AS path information `show ip bgp` command can be used.

`show ip bgp regexp line` [Command]

This commands display BGP routes that matches AS path regular expression *line*.

9.7.3 AS Path Access List

AS path access list is user defined AS path.

`ip as-path access-list word {permit|deny} line` [Command]

This command defines a new AS path access list.

`no ip as-path access-list word` [Command]
`no ip as-path access-list word {permit|deny} line` [Command]

9.7.4 Using AS Path in Route Map

`match as-path word` [Route Map]

`set as-path prepend as-path` [Route Map]

9.7.5 Private AS Numbers

9.8 BGP Communities Attribute

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 *RFC1997, BGP Communities Attribute* and *RFC1998, An Application of the BGP Community Attribute in Multi-home Routing*. It is an optional transitive attribute, therefore local policy can travel through different autonomous system.

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

`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.

When BGP communities attribute is received, duplicated communities value in the communities attribute is ignored and each communities values are sorted in numerical order.

9.8.1 BGP Community Lists

BGP community list is a user defined BGP communites attribute list. BGP community list can be used for matching or manipulating BGP communities attribute in updates.

There are two types of community list. One is standard community list and another is expanded community list. Standard community list defines communities attribute. Expanded community list defines communities attribute string with regular expression. Standard community list is compiled into binary format when user define it. Standard community list will be directly compared to BGP communities attribute in BGP updates. Therefore the comparison is faster than expanded community list.

`ip community-list standard` *name* `{permit|deny}` *community* [Command]

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}` *line* [Command]

This command defines a new expanded community list. *line* is a string expression of communities attribute. *line* can include regular expression to match communities attribute in BGP updates.

`no ip community-list` *name* [Command]
`no ip community-list standard` *name* [Command]
`no ip community-list expanded` *name* [Command]

These commands delete community lists specified by *name*. All of community lists shares a single name space. So community lists can be removed simpley specifying community lists name.

`show ip community-list` [Command]
`show ip community-list` *name* [Command]

This command display current 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
```

9.8.2 Numbered BGP 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`* [Command]

This command defines a new community list. <1-99> is standard community list number. Community list name within this range defines standard community list. When *community* is empty it matches to any routes.

`ip community-list <100-199> {permit|deny} `*`community`* [Command]

This command defines a new community list. <100-199> is expanded community list number. Community list name within this range defines expanded community list.

`ip community-list `*`name`*` {permit|deny} `*`community`* [Command]

When community list type is not specifed, 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 compability. Use of this feature is not recommended.

9.8.3 BGP Community in Route Map

In Route Map (see Chapter 13 [Route Map], page 89), we can match or set BGP communities attribute. Using this feature network operator can implement their network policy based on BGP communities attribute.

Following commands can be used in Route Map.

`match community `*`word`* [Route Map]
`match community `*`word`*` exact-match` [Route Map]

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 spcified, match happen only when BGP updates have completely same communities value specified in the community list.

`set community none` [Route Map]
`set community `*`community`* [Route Map]
`set community `*`community`*` additive` [Route Map]

This command manipulate communities value in BGP updates. When `none` is specified as communities value, it removes entire communities attribute from BGP updates. When *community* is not `none`, specified communities value is set to BGP updates. If BGP updates already has BGP communities value, the existing BGP communities value is replaced with specified *community* value. When `additive` keyword is specified, *community* is appended to the existing communities value.

set comm-list *word* delete [Route Map]

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.

9.8.4 Display BGP Routes by Community

To show BGP routes which has specific BGP communities attribute, `show ip bgp` command can be used. The *community* value and community list can be used for `show ip bgp` command.

show ip bgp community [Command]
show ip bgp community *community* [Command]
show ip bgp community *community* exact-match [Command]

`show ip bgp community` displays BGP routes which has communities attribute. When *community* is specified, BGP routes that matches *community* value is displayed. For this command, `internet` keyword can't be used for *community* value. When `exact-match` is specified, it display only routes that have an exact match.

show ip bgp community-list *word* [Command]
show ip bgp community-list *word* exact-match [Command]

This commands display BGP routes that matches community list *word*. When `exact-match` is specified, display only routes that have an exact match.

9.8.5 Using BGP Communities Attribute

Following configuration is the most typical usage of BGP communities attribute. AS 7675 provides upstream Internet connection to AS 100. When following configuration exists in AS 7675, AS 100 networks operator 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
 neighbor 192.168.0.1 route-map RMAP in
!
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
```

Following configuration announce 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, announced route's local preference will be set to value 80.

```
router bgp 100
 network 10.0.0.0/8
 neighbor 192.168.0.2 remote-as 7675
 neighbor 192.168.0.2 route-map RMAP out
!
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
```

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. Network operator can put special internal communities value at BGP border router, then limit the BGP routes announcement into the internal network.

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

Following exmaple filter BGP routes which has communities value 1:1. When there is no match community-list returns deny. To avoid filtering all of routes, we need to define permit any at last.

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

Communities value keyword `internet` has special meanings in standard community lists. In below example `internet` act as match any. It matches all of BGP routes even if the route does not have communities attribute at all. So community list `INTERNET` is same as above example's `FILTER`.

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

Following configuration is an example of communities value deletion. With this configuration communities value 100:1 and 100:2 is 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
 neighbor 192.168.0.1 route-map RMAP in
!
ip community-list standard DEL permit 100:1 100:2
!
route-map RMAP permit 10
 set comm-list DEL delete
```

9.9 BGP 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. `10.0.0.1:100` represents

9.9.1 BGP Extended Community Lists

Expanded Community Lists is a user defined BGP Expanded Community Lists.

`ip extcommunity-list standard` *name* `{permit|deny}` *extcommunity* [Command]

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* [Command]

This command defines a new expanded extcommunity-list. *line* is a string expression of extended communities attribute. *line* can include regular expression to match extended communities attribute in BGP updates.

`no ip extcommunity-list` *name* [Command]
`no ip extcommunity-list standard` *name* [Command]
`no ip extcommunity-list expanded` *name* [Command]

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 simpley specifying the name.

`show ip extcommunity-list` [Command]
`show ip extcommunity-list` *name* [Command]

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

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

9.9.2 BGP Extended Communities in Route Map

`match extcommunity` *word* [Route Map]

`set extcommunity rt` *extcommunity* [Route Map]

This command set Route Target value.

`set extcommunity soo` *extcommunity* [Route Map]

This command set Site of Origin value.

9.10 Displaying BGP Routes

9.10.1 Show IP BGP

`show ip bgp` [Command]
`show ip bgp` *A.B.C.D* [Command]
`show ip bgp` *X:X::X:X* [Command]

This command displays BGP routes. When no route is specified it display all of IPv4 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
```

9.10.2 More Show IP BGP

`show ip bgp regexp` *line* [Command]

This command display BGP routes using AS path regular expression (see Section 9.7.2 [Display BGP Routes by AS Path], page 51).

`show ip bgp community` *community* [Command]
`show ip bgp community` *community* `exact-match` [Command]

This command display BGP routes using *community* (see Section 9.8.4 [Display BGP Routes by Community], page 55).

`show ip bgp community-list` *word* [Command]
`show ip bgp community-list` *word* `exact-match` [Command]

This command display BGP routes using community list (see Section 9.8.4 [Display BGP Routes by Community], page 55).

`show ip bgp summary` [Command]

`show ip bgp neighbor [`*peer*`]` [Command]

`clear ip bgp` *peer* [Command]

Clear peers which have addresses of X.X.X.X

`clear ip bgp` *peer* `soft in` [Command]

Clear peer using soft reconfiguration.

`show debug` [Command]

`debug event` [Command]

`debug update` [Command]

`debug keepalive` [Command]

`no debug event` [Command]

`no debug update` [Command]

`no debug keepalive` [Command]

9.11 Capability Negotiation

When adding IPv6 routing information exchange feature to BGP. There were some proposals. IETF (Internet Engineering Task Force) IDR (Inter Domain Routing) WG (Working group) adopted a proposal called Multiprotocol Extension for BGP. The specification is described in *RFC2283*. 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 remote peer supports the protocol, `bgpd` can exchange IPv6 and/or multicast routing information.

Traditional BGP did not have the feature to detect 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 to operational network. *RFC2842, Capabilities Advertisement with BGP-4* adopted a feature called Capability Negotiation. `bgpd` use this Capability Negotiation to detect the remote peer's capabilities. If the peer is only configured as IPv4 unicast neighbor, `bgpd` does not send these Capability Negotiation packets (at least not unless other optional BGP features require capability negotation).

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

If you want to completely match capabilities with remote peer. Please use `strict-capability-match` command.

neighbor *peer* strict-capability-match [BGP]
no neighbor *peer* strict-capability-match [BGP]

> 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 [BGP]
no neighbor *peer* dont-capability-negotiate [BGP]

> 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` [BGP]
`no neighbor` *peer* `override-capability` [BGP]

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

9.12 Route Reflector

`bgp cluster-id` *a.b.c.d* [BGP]

`neighbor` *peer* `route-reflector-client` [BGP]
`no neighbor` *peer* `route-reflector-client` [BGP]

9.13 Route Server

At an Internet Exchange point, many ISPs are connected to each other by external BGP peering. Normally these external BGP connection are done by '`full mesh`' method. As with internal BGP full mesh formation, this method has a scaling problem.

This scaling problem is well known. Route Server is a method to resolve the problem. Each ISP's BGP router only peers to Route Server. Route Server serves as BGP information exchange to other BGP routers. By applying this method, numbers of BGP connections is reduced from O(n*(n-1)/2) to O(n).

Unlike normal BGP router, Route Server must have several routing tables for managing different routing policies for each BGP speaker. We call the routing tables as different `views`. `bgpd` can work as normal BGP router or Route Server or both at the same time.

9.13.1 Multiple instance

To enable multiple view function of `bgpd`, you must turn on multiple instance feature beforehand.

`bgp multiple-instance` [Command]

Enable BGP multiple instance feature. After this feature is enabled, you can make multiple BGP instances or multiple BGP views.

`no bgp multiple-instance` [Command]

Disable BGP multiple instance feature. You can not disable this feature when BGP multiple instances or views exist.

When you want to make configuration more Cisco like one,

`bgp config-type cisco` [Command]

Cisco compatible BGP configuration output.

When bgp config-type cisco is specified,

"no synchronization" is displayed. "no auto-summary" is desplayed.

"network" and "aggregate-address" argument is displayed as "A.B.C.D M.M.M.M"

Quagga: network 10.0.0.0/8 Cisco: network 10.0.0.0

Quagga: 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 there is no configuration is specified community attribute and extended community attribute are sent to neighbor. When user manually disable the feature community attribute is not sent to the neighbor. In case of `bgp config-type cisco` is specified, community attribute is not sent to the neighbor by default. To send community attribute user has to specify `neighbor A.B.C.D send-community` command.

```
!
router bgp 1
 neighbor 10.0.0.1 remote-as 1
 no neighbor 10.0.0.1 send-community
!
router bgp 1
 neighbor 10.0.0.1 remote-as 1
 neighbor 10.0.0.1 send-community
!
```

`bgp config-type zebra` [Command]

Quagga style BGP configuration. This is default.

9.13.2 BGP instance and view

BGP instance is a normal BGP process. The result of route selection goes to the kernel routing table. You can setup different AS at the same time when BGP multiple instance feature is enabled.

`router bgp` *`as-number`* [Command]

Make a new BGP instance. You can use arbitrary word for the *name*.

```
bgp multiple-instance
!
router bgp 1
 neighbor 10.0.0.1 remote-as 2
 neighbor 10.0.0.2 remote-as 3
!
router bgp 2
 neighbor 10.0.0.3 remote-as 4
 neighbor 10.0.0.4 remote-as 5
```

BGP view is almost same as normal BGP process. The result of route selection does not go to the kernel routing table. BGP view is only for exchanging BGP routing information.

`router bgp` *`as-number`* `view` *`name`* [Command]

Make a new BGP view. You can use arbitrary word for the *name*. This view's route selection result does not go to the kernel routing table.

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

```
bgp multiple-instance
!
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
```

9.13.3 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
 neighbor 10.0.0.1 distribute-list 1 in
!
router bgp 1 view 2
 neighbor 10.0.0.1 remote-as 2
 neighbor 10.0.0.1 distribute-list 2 in
```

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.

9.13.4 Viewing the view

To display routing table of BGP view, you must specify view name.

`show ip bgp view` *name* [Command]

Display routing table of BGP view *name*.

9.14 How to set up a 6-Bone connection

```
zebra configuration
===================
!
! Actually there is no need to configure zebra
!

bgpd configuration
==================
!
! This means that routes go through zebra and into the kernel.
!
router zebra
!
! MP-BGP configuration
!
router bgp 7675
 bgp router-id 10.0.0.1
 neighbor 3ffe:1cfa:0:2:2a0:c9ff:fe9e:f56 remote-as as-number
!
 address-family ipv6
 network 3ffe:506::/32
 neighbor 3ffe:1cfa:0:2:2a0:c9ff:fe9e:f56 activate
 neighbor 3ffe:1cfa:0:2:2a0:c9ff:fe9e:f56 route-map set-nexthop out
 neighbor 3ffe:1cfa:0:2:2c0:4fff:fe68:a231 remote-as as-number
 neighbor 3ffe:1cfa:0:2:2c0:4fff:fe68:a231 route-map set-nexthop out
 exit-address-family
!
ipv6 access-list all permit any
!
! Set output nexthop address.
!
route-map set-nexthop permit 10
 match ipv6 address all
 set ipv6 nexthop global 3ffe:1cfa:0:2:2c0:4fff:fe68:a225
 set ipv6 nexthop local fe80::2c0:4fff:fe68:a225
!
! logfile FILENAME is obsolete.  Please use log file FILENAME

log file bgpd.log
!
```

9.15 Dump BGP packets and table

`dump bgp all` *path* [Command]

`dump bgp all` *path interval* [Command]

Dump all BGP packet and events to *path* file.

`dump bgp updates` *path* [Command]
`dump bgp updates` *path interval* [Command]

Dump BGP updates to *path* file.

`dump bgp routes` *path* [Command]
`dump bgp routes` *path* [Command]

Dump whole BGP routing table to *path*. This is heavy process.

9.16 BGP 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
 network 10.236.87.0/24
 neighbor upstream peer-group
 neighbor upstream remote-as 64515
 neighbor upstream capability dynamic
 neighbor upstream prefix-list pl-allowed-adv out
 neighbor 10.1.1.1 peer-group upstream
 neighbor 10.1.1.1 description ACME ISP
!
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. With upstream, peer and customer sessions. Advertising global prefixes and NO_EXPORT prefixes and providing actions for customer routes based on community values. Extensive use 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 containts silly mistakes, if not serious flaws.

```
router bgp 64512
 bgp router-id 10.236.87.1
 network 10.123.456.0/24
 network 10.123.456.128/25 route-map rm-no-export
 neighbor upstream capability dynamic
 neighbor upstream route-map rm-upstream-out out
 neighbor cust capability dynamic
 neighbor cust route-map rm-cust-in in
 neighbor cust route-map rm-cust-out out
 neighbor cust send-community both
 neighbor peer capability dynamic
 neighbor peer route-map rm-peer-in in
 neighbor peer route-map rm-peer-out out
 neighbor peer send-community both
 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 10.3.1.1 prefix-list pl-cust1-network in
 neighbor 10.4.1.1 remote-as 64518
 neighbor 10.4.1.1 peer-group cust
 neighbor 10.4.1.1 prefix-list pl-cust2-network in
 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 prefix-list pl-peer1-network in
 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 prefix-list pl-peer2-network in
 neighbor 10.6.1.1 description peer AS 2
!
ip prefix-list pl-default permit 0.0.0.0/0
!
ip prefix-list pl-upstream-peers permit 10.1.1.1/32
ip prefix-list pl-upstream-peers permit 10.2.1.1/32
!
ip prefix-list pl-cust1-network permit 10.3.1.0/24
ip prefix-list pl-cust1-network permit 10.3.2.0/24
!
ip prefix-list pl-cust2-network permit 10.4.1.0/24
!
ip prefix-list pl-peer1-network permit 10.5.1.0/24
ip prefix-list pl-peer1-network permit 10.5.2.0/24
ip prefix-list pl-peer1-network permit 192.168.0.0/24
!
ip prefix-list pl-peer2-network permit 10.6.1.0/24
ip prefix-list pl-peer2-network permit 10.6.2.0/24
ip prefix-list pl-peer2-network permit 192.168.1.0/24
ip prefix-list pl-peer2-network permit 192.168.2.0/24
ip prefix-list pl-peer2-network permit 172.16.1/24
!
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
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 cant 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
 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
```

10 Configuring Quagga 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 (see Figure 10.2), we can convert it into a centralized scenario where each of the four establishes a single BGP peering against the Route Server (see Figure 10.3).

We will first describe briefly the Route Server model implemented by Quagga. We will explain the commands that have been added for configuring that model. And finally we will show a full example of Quagga configured as Route Server.

10.1 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 Figure 10.1, 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.

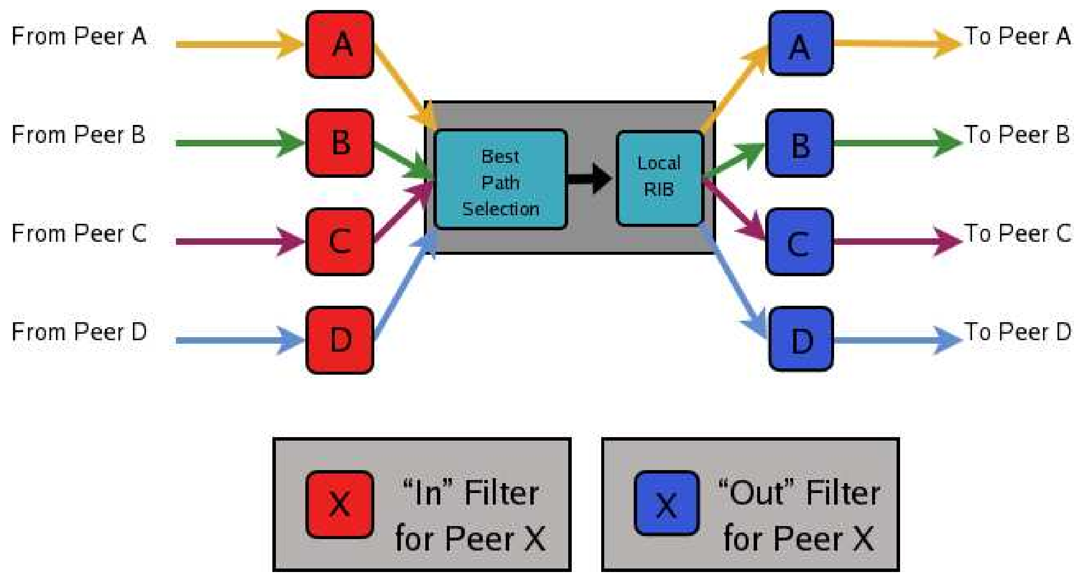

Figure 10.1: Announcement processing inside a "normal" BGP speaker

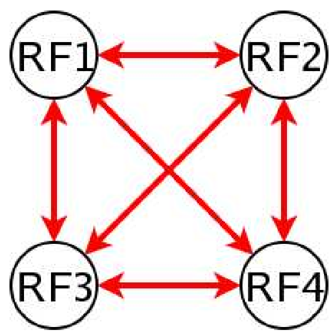

Figure 10.2: Full Mesh

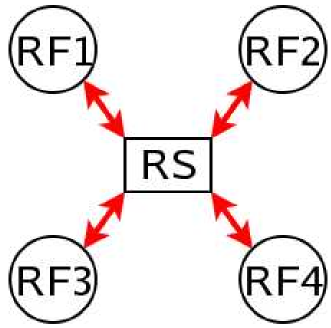

Figure 10.3: Route Server and clients

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 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 *RFC1771*, 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 Quagga is shown in Figure 10.4. 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).

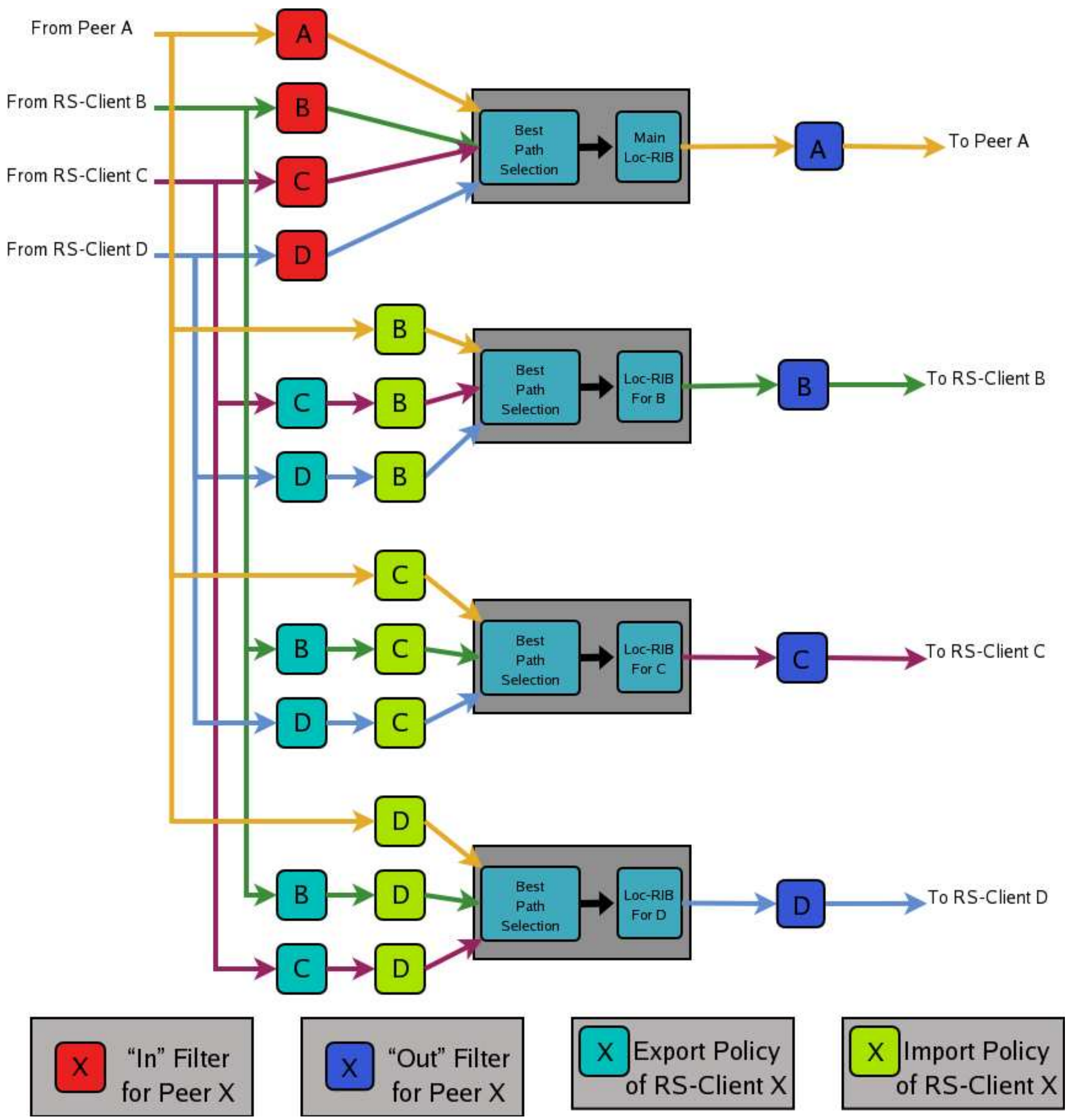

Figure 10.4: Announcement processing model implemented by the Route Server

10.2 Commands for configuring a Route Server

Now we will describe the commands that have been added to quagga in order to support the route server features.

`neighbor` *peer-group* `route-server-client` [Route-Server]

`neighbor A.B.C.D route-server-client` [Route-Server]
`neighbor X:X::X:X route-server-client` [Route-Server]

This command configures the peer given by *peer*, *A.B.C.D* or *X:X::X:X* as an RS-client.

Actually this command is not new, it already existed in standard Quagga. 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 Figure 10.4.). 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}` [Route-Server]

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}` [Route-Server]

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 Figure 10.4.).

`call WORD` [Route-map Command]

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.

10.3 Example of Route Server Configuration

Finally we are going to show how to configure a Quagga 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 VNUML.

10.3.1 Configuration of the BGP routers without Route Server

We will suppose that our initial scenario is an exchange point with three BGP capable routers, named RA, RB and RC. Each of the BGP speakers generates some routes (with the *network* command), and establishes BGP peerings against the other two routers. These peerings have In and Out route-maps configured, named like “PEER-X-IN” or “PEER-X-OUT”. For example the configuration file for router RA could be the following:

```
#Configuration for router 'RA'
!
hostname RA
password ****
!
router bgp 65001
  no bgp default ipv4-unicast
  neighbor 2001:0DB8::B remote-as 65002
  neighbor 2001:0DB8::C remote-as 65003
!
  address-family ipv6
    network 2001:0DB8:AAAA:1::/64
    network 2001:0DB8:AAAA:2::/64
    network 2001:0DB8:0000:1::/64
    network 2001:0DB8:0000:2::/64

    neighbor 2001:0DB8::B activate
    neighbor 2001:0DB8::B soft-reconfiguration inbound
    neighbor 2001:0DB8::B route-map PEER-B-IN in
    neighbor 2001:0DB8::B route-map PEER-B-OUT out

    neighbor 2001:0DB8::C activate
    neighbor 2001:0DB8::C soft-reconfiguration inbound
    neighbor 2001:0DB8::C route-map PEER-C-IN in
    neighbor 2001:0DB8::C route-map PEER-C-OUT out
  exit-address-family
!
ipv6 prefix-list COMMON-PREFIXES seq  5 permit 2001:0DB8:0000::/48 ge 64 le 64
ipv6 prefix-list COMMON-PREFIXES seq 10 deny any
!
ipv6 prefix-list PEER-A-PREFIXES seq  5 permit 2001:0DB8:AAAA::/48 ge 64 le 64
ipv6 prefix-list PEER-A-PREFIXES seq 10 deny any
!
ipv6 prefix-list PEER-B-PREFIXES seq  5 permit 2001:0DB8:BBBB::/48 ge 64 le 64
ipv6 prefix-list PEER-B-PREFIXES seq 10 deny any
!
ipv6 prefix-list PEER-C-PREFIXES seq  5 permit 2001:0DB8:CCCC::/48 ge 64 le 64
ipv6 prefix-list PEER-C-PREFIXES seq 10 deny any
!
route-map PEER-B-IN permit 10
```

```
  match ipv6 address prefix-list COMMON-PREFIXES
  set metric 100
route-map PEER-B-IN permit 20
  match ipv6 address prefix-list PEER-B-PREFIXES
  set community 65001:11111
!
route-map PEER-C-IN permit 10
  match ipv6 address prefix-list COMMON-PREFIXES
  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
!
```

10.3.2 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.

10.3.3 Configuration of the Route Server itself

As we said when we described the functions of a route server (see Section 10.1 [Description of the Route Server model], page 71), 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
    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.

10.3.4 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 posible 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
  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.

11 VTY shell

`vtysh` is integrated shell of Quagga software.

To use vtysh please specify —enable-vtysh to configure script. To use PAM for authentication use —with-libpam option to configure script.

vtysh only searches /etc/quagga path for vtysh.conf which is the vtysh configuration file. Vtysh does not search current directory for configuration file because the file includes user authentication settings.

Currently, vtysh.conf has only two commands.

11.1 VTY shell username

`username` *`username`* `nopassword` [Command]

With this set, user foo does not need password authentication for user vtysh. With PAM vtysh uses PAM authentication mechanism.

If vtysh is compiled without PAM authentication, every user can use vtysh without authentication. vtysh requires read/write permission to the various daemons vty sockets, this can be accomplished through use of unix groups and the –enable-vty-group configure option.

11.2 VTY shell integrated configuration

`service integrated-vtysh-config` [Command]

Write out integrated Quagga.conf file when 'write file' is issued.

This command controls the behaviour of vtysh when it is told to write out the configuration. Per default, vtysh will instruct each daemon to write out their own config files when `write file` is issued. However, if `service integrated-vtysh-config` is set, when `write file` is issued, vtysh will instruct the daemons will write out a Quagga.conf with all daemons' commands integrated into it.

Vtysh per default behaves as if `write-conf daemon` is set. Note that both may be set at same time if one wishes to have both Quagga.conf and daemon specific files written out. Further, note that the daemons are hard-coded to first look for the integrated Quagga.conf file before looking for their own file.

We recommend you do not mix the use of the two types of files. Further, it is better not to use the integrated Quagga.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.

12 Filtering

Quagga 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.

12.1 IP Access List

`access-list` *name* `permit` *ipv4-network* [Command]
`access-list` *name* `deny` *ipv4-network* [Command]

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
```

12.2 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*`]` [Command]
`ip prefix-list` *name* `seq` *number* `(permit|deny)` *prefix* `[le` *len*`] [ge` *len*`]` [Command]

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
: `le` command specifies prefix length. The prefix list will be applied if the prefix length is less than or equal to the le prefix length.

ge
: `ge` command 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* [Command]

12.2.1 ip prefix-list description

`ip prefix-list` *name* `description` *desc* [Command]

Descriptions may be added to prefix lists. This command adds a description to the prefix list.

`no ip prefix-list` *name* `description [`*desc*`]` [Command]

Deletes the description from a prefix list. It is possible to use the command without the full description.

12.2.2 ip prefix-list sequential number control

`ip prefix-list sequence-number` [Command]

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

`no ip prefix-list sequence-number` [Command]

With this command, the IP prefix list sequential number is not displayed.

12.2.3 Showing ip prefix-list

`show ip prefix-list` [Command]

Display all IP prefix lists.

`show ip prefix-list` *name* [Command]

Show IP prefix list can be used with a prefix list name.

`show ip prefix-list` *name* `seq` *num* [Command]

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* [Command]

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` [Command]

`show ip prefix-list` *name a.b.c.d/m* `first-match` [Command]

`show ip prefix-list summary` [Command]

`show ip prefix-list summary` *name* [Command]

`show ip prefix-list detail` [Command]

`show ip prefix-list detail` *name* [Command]

12.2.4 Clear counter of ip prefix-list

`clear ip prefix-list` [Command]

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* [Command]

`clear ip prefix-list` *name a.b.c.d/m* [Command]

13 Route Map

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 distincts sets of clauses:

'`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 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.

'`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 explicitely 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.

'`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

13.1 Route Map Command

route-map *route-map-name* (permit|deny) *order* [Command]

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

13.2 Route Map Match Command

match ip address *access_list* [Route-map Command]

Matches the specified *access_list*

match ip next-hop *ipv4_addr* [Route-map Command]

Matches the specified *ipv4_addr*.

match aspath *as_path* [Route-map Command]

Matches the specified *as_path*.

match metric *metric* [Route-map Command]

Matches the specified *metric*.

match community *community_list* [Route-map Command]

Matches the specified *community_list*

13.3 Route Map Set Command

`set ip next-hop` *`ipv4_address`* [Route-map Command]
: Set the BGP nexthop address.

`set local-preference` *`local_pref`* [Route-map Command]
: Set the BGP local preference.

`set weight` *`weight`* [Route-map Command]
: Set the route's weight.

`set metric` *`metric`* [Route-map Command]
: Set the BGP attribute MED.

`set as-path prepend` *`as_path`* [Route-map Command]
: Set the BGP AS path to prepend.

`set community` *`community`* [Route-map Command]
: Set the BGP community attribute.

`set ipv6 next-hop global` *`ipv6_address`* [Route-map Command]
: Set the BGP-4+ global IPv6 nexthop address.

`set ipv6 next-hop local` *`ipv6_address`* [Route-map Command]
: Set the BGP-4+ link local IPv6 nexthop address.

13.4 Route Map Call Command

`call` *`name`* [Route-map Command]
: Call route-map *name*. If it returns deny, deny the route and finish processing the route-map.

13.5 Route Map Exit Action Command

`on-match next` [Route-map Command]
`continue` [Route-map Command]
: Proceed on to the next entry in the route-map.

`on-match goto` *`N`* [Route-map Command]
`continue` *`N`* [Route-map Command]
: Proceed processing the route-map at the first entry whose order is >= N

13.6 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 Section 9.16 [BGP Configuration Examples], page 65 for examples of more sophisticated useage of route-maps, including of the '`call`' action.

14 IPv6 Support

Quagga fully supports IPv6 routing. As described so far, Quagga supports RIPng, OSPFv3 and BGP-4+. You can give IPv6 addresses to an interface and configure static IPv6 routing information. Quagga 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.

14.1 Router Advertisement

`no ipv6 nd suppress-ra` [Interface Command]

Send router advertisment messages.

`ipv6 nd suppress-ra` [Interface Command]

Don't send router advertisment messages.

`ipv6 nd prefix` *`ipv6prefix`* `[`*`valid-lifetime`*`]` `[`*`preferred-lifetime`*`] [off-link] [no-autoconfig] [router-address]` [Interface Command]

Configuring the IPv6 prefix to include in router advertisements. Several prefix specific optional parameters and flags may follow:

- *valid-lifetime* - the length of time in seconds during what the prefix is valid for the purpose of on-link determination. Value *infinite* represents infinity (i.e. a value of all one bits (`0xffffffff`)).

  Range: `<0-4294967295>` Default: `2592000`
- *preferred-lifetime* - the length of time in seconds during what addresses generated from the prefix remain preferred. Value *infinite* represents infinity.

  Range: `<0-4294967295>` Default: `604800`
- *off-link* - indicates that advertisement makes no statement about on-link or off-link properties of the prefix.

  Default: not set, i.e. this prefix can be used for on-link determination.
- *no-autoconfig* - indicates to hosts on the local link that the specified prefix cannot be used for IPv6 autoconfiguration.

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

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

`ipv6 nd ra-interval SECONDS` [Interface Command]
`no ipv6 nd ra-interval` [Interface Command]

The maximum time allowed between sending unsolicited multicast router advertisements from the interface, in seconds. Must be no less than 3 seconds.

Default: `600`

`ipv6 nd ra-interval msec MILLISECONDS` [Interface Command]
`no ipv6 nd ra-interval msec` [Interface Command]

The maximum time allowed between sending unsolicited multicast router advertisements from the interface, in milliseconds. Must be no less than 30 milliseconds.

Default: `600000`

`ipv6 nd ra-lifetime SECONDS` [Interface Command]
`no ipv6 nd ra-lifetime` [Interface Command]

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

Default: `1800`

`ipv6 nd reachable-time MILLISECONDS` [Interface Command]
`no ipv6 nd reachable-time` [Interface Command]

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). Must be no greater than `3,600,000` milliseconds (1 hour).

Default: `0`

`ipv6 nd managed-config-flag` [Interface Command]
`no ipv6 nd managed-config-flag` [Interface Command]

Set/unset flag in IPv6 router advertisements which indicates to hosts that they should use managed (stateful) protocol for addresses autoconfiguration in addition to any addresses autoconfigured using stateless address autoconfiguration.

Default: not set

`ipv6 nd other-config-flag` [Interface Command]
`no ipv6 nd other-config-flag` [Interface Command]

Set/unset flag in IPv6 router advertisements which indicates to hosts that they should use administered (stateful) protocol to obtain autoconfiguration information other than addresses.

Default: not set

`ipv6 nd home-agent-config-flag` [Interface Command]
`no ipv6 nd home-agent-config-flag` [Interface Command]

Set/unset flag in IPv6 router advertisements which indicates to hosts that the router acts as a Home Agent and includes a Home Agent Option.

Default: not set

`ipv6 nd home-agent-preference` [Interface Command]
`no ipv6 nd home-agent-preference` [Interface Command]

The value to be placed in Home Agent Option, when Home Agent config flag is set, which indicates to hosts Home Agent preference.

Default: 0

`ipv6 nd home-agent-lifetime` [Interface Command]
`no ipv6 nd home-agent-lifetime` [Interface Command]

The value to be placed in Home Agent Option, when Home Agent config flag is set, which indicates to hosts Home Agent Lifetime. A value of 0 means to place Router Lifetime value.

Default: 0

`ipv6 nd adv-interval-option` [Interface Command]
`no ipv6 nd adv-interval-option` [Interface Command]

Include an Advertisement Interval option which indicates to hosts the maximum time, in milliseconds, between successive unsolicited Router Advertisements.

Default: not set

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

For more information see *RFC2462 (IPv6 Stateless Address Autoconfiguration)* , *RFC2461 (Neighbor Discovery for IP Version 6 (IPv6))* and *RFC3775 (Mobility Support in IPv6 (Mobile IPv6))*.

15 Kernel Interface

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

'`ioctl`' The '`ioctl`' 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`' '`sysctl`' 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`'
'`proc filesystem`' 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 Quagga 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 Quagga. You can use netlink as a dynamic routing update channel between Quagga and the kernel.

16 SNMP Support

SNMP (Simple Network Managing Protocol) is a widely implemented feature for collecting network information from router and/or host. Quagga itself does not support SNMP agent (server daemon) functionality but is able to connect to a SNMP agent using the SMUX protocol (*RFC1227*) and make the routing protocol MIBs available through it.

16.1 Getting and installing an SNMP agent

There are several SNMP agent which support SMUX. 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. `net-snmp` has to be compiled with `--with-mib-modules=smux` to be able to accept connections from Quagga.

16.2 SMUX configuration

To enable SMUX protocol support, Quagga must have been build with the `--enable-snmp` option.

A separate connection has then to be established between between the SNMP agent (snmpd) and each of the Quagga daemons. This connections each use different OID numbers and passwords. Be aware that this OID number is not the one that is used in queries by clients, it is solely used for the intercommunication of the daemons.

In the following example the ospfd daemon will be connected to the snmpd daemon using the password "quagga_ospfd". For testing it is recommending to take exactly the below snmpd.conf as wrong access restrictions can be hard to debug.

```
/etc/snmp/snmpd.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
#
# the following line is relevant for Quagga
#
smuxpeer .1.3.6.1.4.1.3317.1.2.5 quagga_ospfd

/etc/quagga/ospf:
! ... the rest of ospfd.conf has been omitted for clarity ...
!
smux peer .1.3.6.1.4.1.3317.1.2.5 quagga_ospfd
!
```

After restarting snmpd and quagga, a successful connection can be verified in the syslog and by querying the SNMP daemon:

```
snmpd[12300]: [smux_accept] accepted fd 12 from 127.0.0.1:36255
snmpd[12300]: accepted smux peer: \
oid GNOME-PRODUCT-ZEBRA-MIB::ospfd, quagga-0.96.5

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

Be warned that the current version (5.1.1) of the Net-SNMP daemon writes a line for every SNMP connect to the syslog which can lead to enormous log file sizes. If that is a problem you should consider to patch snmpd and comment out the troublesome `snmp_log()` line in the function `netsnmp_agent_check_packet()` in `agent/snmp_agent.c`.

16.3 MIB and command reference

The following OID numbers are used for the interprocess communication of snmpd and the Quagga daemons. Sadly, SNMP has not been implemented in all daemons yet.

```
            (OIDs below .iso.org.dod.internet.private.enterprises)
zebra .1.3.6.1.4.1.3317.1.2.1 .gnome.gnomeProducts.zebra.zserv
bgpd .1.3.6.1.4.1.3317.1.2.2 .gnome.gnomeProducts.zebra.bgpd
ripd .1.3.6.1.4.1.3317.1.2.3 .gnome.gnomeProducts.zebra.ripd
ospfd .1.3.6.1.4.1.3317.1.2.5 .gnome.gnomeProducts.zebra.ospfd
ospf6d .1.3.6.1.4.1.3317.1.2.6 .gnome.gnomeProducts.zebra.ospf6d
```

The following OID numbers are used for querying the SNMP daemon by a client:

```
zebra .1.3.6.1.2.1.4.24   .iso.org.dot.internet.mgmt.mib-2.ip.ipForward
ospfd .1.3.6.1.2.1.14     .iso.org.dot.internet.mgmt.mib-2.ospf
bgpd .1.3.6.1.2.1.15     .iso.org.dot.internet.mgmt.mib-2.bgp
ripd .1.3.6.1.2.1.23     .iso.org.dot.internet.mgmt.mib-2.rip2
ospf6d .1.3.6.1.3.102     .iso.org.dod.internet.experimental.ospfv3
```

The following syntax is understood by the Quagga daemons for configuring SNMP:

`smux peer` *oid* [Command]
`no smux peer` *oid* [Command]

`smux peer` *oid password* [Command]
`no smux peer` *oid password* [Command]

16.4 Handling SNMP Traps

To handle snmp traps make sure your snmp setup of quagga works correctly as described in the quagga documentation in See Chapter 16 [SNMP Support], page 99.

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
asdescr=`echo "$WHOISINFO" | grep "^descr:" | sed -e 's/^descr://g' -e 's/  //g' -e 's/^ //g'
```

```
# 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" ;;
  *) 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 "$suberrorcode" in
      01) suberror="Malformed Attribute List" ;;
      02) suberror="Unrecognized Well-known Attribute" ;;
      03) suberror="Missing Well-known Attribute" ;;
      04) suberror="Attribute Flags Error" ;;
      05) suberror="Attribute Length Error" ;;
      06) suberror="Invalid ORIGIN Attribute" ;;
      07) suberror="AS Routing Loop" ;;
      08) suberror="Invalid NEXT_HOP Attribute" ;;
      09) suberror="Optional Attribute Error" ;;
      10) suberror="Invalid Network Field" ;;
      11) suberror="Malformed AS_PATH" ;;
      *) suberror="Unknown" ;;
    esac
    ;;
  04)
    error="Hold Timer Expired"
    suberror=""
    ;;
  05)
    error="Finite State Machine Error"
    suberror=""
    ;;
  06)
    error="Cease"
    case "$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
```

Appendix A Zebra Protocol

A.1 Overview of the Zebra Protocol

Zebra Protocol is used by protocol daemons to communicate with the zebra daemon.

Each protocol daemon may request and send information to and from the zebra daemon such as interface states, routing state, nexthop-validation, and so on. Protocol daemons may also install routes with zebra. The zebra daemon manages which route is installed into the forwarding table with the kernel.

Zebra Protocol is a streaming protocol, with a common header. Two versions of the header are in use. Version 0 is implicitely versioned. Version 1 has an explicit version field. Version 0 can be distinguished from all other versions by examining the 3rd byte of the header, which contains a marker value for all versions bar version 0. The marker byte corresponds to the command field in version 0, and the marker value is a reserved command in version 0.

We do not anticipate there will be further versions of the header for the foreseeable future, as the command field in version 1 is wide enough to allow for future extensions to done compatibly through seperate commands.

Version 0 is used by all versions of GNU Zebra as of this writing, and versions of Quagga up to and including Quagga 0.98. Version 1 will be used as of Quagga 1.0.

A.2 Zebra Protocol Definition

A.2.1 Zebra Protocol Header (version 0)

```
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
+-------------------------------+---------------+
|           Length (2)          |   Command (1) |
+-------------------------------+---------------+
```

A.2.2 Zebra Protocol Common Header (version 1)

```
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
+-------------------------------+---------------+-------------+
|           Length (2)          |   Marker (1)  | Version (1) |
+-------------------------------+---------------+-------------+
|          Command (2)          |
+-------------------------------+
```

A.2.3 Zebra Protocol Header Field Definitions

'`Length`' Total packet length including this header. The minimum length is 3 bytes for version 0 messages and 6 bytes for version 1 messages.

'`Marker`' Static marker with a value of 255 always. This is to allow version 0 Zserv headers (which do not include version explicitely) to be distinguished from versioned headers. Not present in version 0 messages.

‘`Version`’ Version number of the Zserv message. Clients should not continue processing messages past the version field for versions they do not recognise. Not present in version 0 messages.

‘`Command`’ The Zebra Protocol command.

A.2.4 Zebra Protocol Commands

Command	**Value**
ZEBRA_INTERFACE_ADD	1
ZEBRA_INTERFACE_DELETE	2
ZEBRA_INTERFACE_ADDRESS_ADD	3
ZEBRA_INTERFACE_ADDRESS_DELETE	4
ZEBRA_INTERFACE_UP	5
ZEBRA_INTERFACE_DOWN	6
ZEBRA_IPV4_ROUTE_ADD	7
ZEBRA_IPV4_ROUTE_DELETE	8
ZEBRA_IPV6_ROUTE_ADD	9
ZEBRA_IPV6_ROUTE_DELETE	10
ZEBRA_REDISTRIBUTE_ADD	11
ZEBRA_REDISTRIBUTE_DELETE	12
ZEBRA_REDISTRIBUTE_DEFAULT_ADD	13
ZEBRA_REDISTRIBUTE_DEFAULT_DELETE	14
ZEBRA_IPV4_NEXTHOP_LOOKUP	15
ZEBRA_IPV6_NEXTHOP_LOOKUP	16

Appendix B Packet Binary Dump Format

Quagga can dump routing protocol packet into file with a binary format (see Section 9.15 [Dump BGP packets and table], page 64).

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.

```
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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                              Time                             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|             Type              |            Subtype            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                             Length                            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

If 'type' is PROTOCOL_BGP4MP, 'subtype' is BGP4MP_STATE_CHANGE, 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|        Source AS number       |     Destination AS number     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|        Interface Index        |       Address Family          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       Source IP address                       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                     Destination IP address                    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|            Old State          |           New State           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Where State is the value defined in RFC1771.

If 'type' is PROTOCOL_BGP4MP, 'subtype' is BGP4MP_STATE_CHANGE, 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)          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|             Old State         |             New State         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

If ‘type’ is PROTOCOL_BGP4MP, ‘subtype’ is BGP4MP_MESSAGE, 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         Source AS number      |     Destination AS number     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         Interface Index       |       Address Family          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         Source IP address                     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                      Destination IP address                   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        BGP Message Packet                     |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

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

```
 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                       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        Next Hop Address (Cont'd)              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        Next Hop Address (Cont'd)              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        Next Hop Address (Cont'd)              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Prefix Length |             Address Prefix [variable]         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Address Prefix (cont'd) [variable]        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|       Attribute Length        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|      BGP Attribute [variable length]         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

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,

```
 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 #              |       File Name [variable]    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

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
  /* subtype value */
  #define BGP4MP_STATE_CHANGE 0
  #define BGP4MP_MESSAGE 1
  #define BGP4MP_ENTRY 2
  #define BGP4MP_SNAPSHOT 3
```

Command Index

A

B

C

D

L

M

N

O

P

R

S

T

U

V

W

VTY Key Index

?

C

D

L

M

R

T

U

Index