Project Number:	**IST-2001-32603**
Project Title:	**6NET**
CEC Deliverable Number:	**32603/WWU(JOIN)/DS/3.1.2v2/A1**
Contractual Date of Delivery to the CEC:	30th November 2004
Actual Date of Delivery to the CEC:	
Title of Deliverable:	IPv6 cookbook for routing, DNS, intra-domain multicast, inter-domain multicast, security
Work package contributing to Deliverable:	3
Type of Deliverable*:	R
Deliverable Security Class**:	PU
Editors:	Tina Strauf, Christian Schild
Contributors:	Gunter Van De Velde, Jan Novak, Jerome Durand, Stig Venaas, Christian Schild, Tina Strauf, Carlos Friaças, János Mohácsi.
Reviewers	Mónica Domingues, Carlos Friaças

* Type: P - Prototype, R - Report, D - Demonstrator, O - Other

** Security Class: PU- Public, PP – Restricted to other programme participants (including the Commission), RE – Restricted to a group defined by the consortium (including the Commission), CO – Confidential, only for members of the consortium (including the Commission)

Abstract:

In many ways the issues IPv6 routing, IPv6 DNS, IPv6 multicast and security are very similar to the corresponding tasks in IPv4, which the authors of this document assumes the reader is familiar with. This deliverable rather focuses on the differences between IPv6 and IPv4 in these areas and therefore builds on the IPv4 knowledge and understanding of the reader to enable him to extend or migrate his network to IPv6.

Keywords:

IPv6, Routing, BGP, RIP, OSPFv3, IS-IS, DNS, Bind, multicast, security

Table of Contents

1 Introduction

In many ways the issues IPv6 routing, IPv6 DNS, IPv6 multicast and security are very similar to the corresponding tasks in IPv4, which the authors of this document assumes the reader is familiar with. This deliverable rather focuses on the differences between IPv6 and IPv4 in these areas both with regards to the theory behind certain features nd actual configuration examples.

2 Router Configuration in the Context of Neighbour Discovery

Neighbor discovery is a protocol that allows different nodes on the same link to advertise their existence to their neighbors, and to learn about the existence of their neighbors. It is a basic functionality all implementations of IPv6 on any platform must include.

In the context of router configuration Neighbour Discovery along as Router Discovery has a few special implications. A router periodically sends a router advertisement from each of its multicast interfaces, announcing its availability. Hosts listen for these advertisements for address autoconfiguration and discovery of link-local addresses of the neighboring routers. When a host starts, it multicasts a router solicitation to ask for immediate advertisements.

The router discovery messages do not constitute a routing protocol. They enable hosts to discover the existence of neighboring routers, but are not used to determine which router is best to reach a particular destination. If more than one router advertises its presence on the link, it depends on the IPv6 implementation of the host how it decides which router to use as its preferred default route.

Neighbor discovery uses the following Internet Control Message Protocol Version 6 (ICMPv6) messages: router solicitation, router advertisement, neighbor solicitation, neighbor advertisement, and redirect.

Neighbor discovery for IPv6 replaces the following IPv4 protocols: router discovery (RDISC), Address Resolution Protocol (ARP) and ICMPv4 redirect.

Neighbor discovery is defined in the following documents:

- RFC 2461, Neighbor Discovery for IP Version 6;
- RFC 2462, IPv6 Stateless Address Autoconfiguration;
- RFC 2463, Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 Specification.

RFC 2461 and 2462 are currently in the process of being revised by the IPv6 working group of the IETF. These drafts [RFC2461bis][RFC2462bis] will eventually replace the older RFCs.

Router advertisements can contain a list of prefixes. These prefixes are used for stateless address autoconfiguration of hosts on the link and to maintain a database of on-link (on the same data link) prefixes as well as for duplicate address detection. For routers the list of prefixes, which are on-link is used for forwarding decisions. If the destination address in an IP packet belongs to an on-link prefix, the router forwards the packets to that node. If the node is not on-link, the packets are sent to the next router for consideration. For IPv6 router advertisements, each prefix in the prefix list can contain a prefix length, valid lifetime for the prefix, preferred lifetime for the prefix, an on-link flag and an autoconfiguration flag. This information enables address autoconfiguration and the setting of link parameters such as maximum transmission unit (MTU) size and hop limit.

To summarize, router advertisement messages typically include the following information:

- One or more on-link IPv6 prefixes that nodes on the local link can use to automatically configure their IPv6 addresses;
- Lifetime information for each prefix included in the advertisement;
- Sets of flags that indicate the type of autoconfiguration (stateless or statefull) that can be completed;

- Default router information (whether the router sending the advertisement should be used as a default router and, if so, the amount of time (in seconds) the router should be used as a default router);
- Additional information for hosts, such as the hop limit and MTU a host should use in packets that it originates.

2.1 Cisco IOS

Cisco IOS allows the configuration of the following router parameters in router advertisements:

- The time interval between periodic router advertisement messages;
- The "router lifetime" value, which indicates the usefulness of a router as the default router (for use by all nodes on a given link);
- The network prefixes in use on a given link;
- The time interval between neighbor solicitation message retransmissions (on a given link);
- The amount of time a node considers a neighbor reachable (for use by all nodes on a given link).

2.1.1 Enabling Router Advertisement Messages

The configured parameters are specific to an interface. The sending of router advertisement messages (with default values) is automatically enabled on Ethernet and FDDI interfaces when the **`ipv6 unicast-routing`** global configuration command is configured. This means that a router will automatically send out router advertisements for the prefix corresponding to the (global) addresses configured on an interface belonging to the particular link. That is, if there is a statement of the form:

```
Router (config-if)# ipv6 address address/prefix-length
```

Within the interface configuration context the router will automatically send router advertisements for the prefix this address was built from. However, for stateless autoconfiguration to work properly, the advertised prefix length in router advertisement messages must always be 64 bits.

For other interface types, the sending of router advertisement messages must be manually configured by using the **`no ipv6 nd suppress-ra`** global configuration command. The sending of router advertisement messages can be disabled on individual interfaces by using the **`ipv6 nd suppress-ra`** interface configuration command.

2.1.1.1 Example

If an interface is configured in the following way, the router will automatically send out router advertisements for the prefix `2001:638:500:101::/64` on the link this interface is attached to.

```
interface Ethernet0/0
 ipv6 address 2001:638:500:101::/64 eui-64
```

If the interface was additionally configured as follows, the prefix list in router advertisements on this link would include prefix `2001:638:102::/64`.

```
ipv6 address 2001:638:500:102:aaaa:bbbb:cccc:dddd/64
```

2.1.2 Customizing Prefix Information in Router Advertisement Messages

Aside from the prefixes to be advertised on a link Cisco IOS allows the configuration of the following router parameters in router advertisements:

- If more than one address is configured on an interface the network prefixes to be advertised on a given link. It is also possible to advertise prefixes not configured on the interface itself .
- The time interval between periodic router advertisement messages. This parameter is set for the interface in general. That means all prefixes to be advertised on the interface will be advertised at the same time albeit possibly with different lifetimes.
- The "router lifetime" value, which indicates the usefulness of a router as the default router (for use by all nodes on a given link). This means that routers can theoretically advertise prefixes for use with stateless address autoconfiguration while not acting as default router for hosts on.
- The time interval between neighbor solicitation message retransmissions (on a given link).
- The amount of time a node considers a neighbour reachable (for use by all nodes on a given link).

Command Syntax:

```
Router (config-if)# ipv6 nd prefix ipv6-prefix/prefix-length \\
   | default [[valid-lifetime preferred-lifetime] \\
   | [at valid-date preferred-date] | infinite \\
   | no-advertise | off-link | no-autoconfig]]
```

This command allows control over the individual parameters per prefix, including whether or not the prefix should be advertised at all.

As mentioned above, prefixes configured as addresses on an interface using the **`ipv6 address`** command are advertised in router advertisements by default. However, if you configure prefixes for advertisement using the **`ipv6 nd prefix`** command, then only these prefixes are advertised.

The **`default`** keyword can be used to set default parameters for all prefixes.

With the **`at`** `valid-date preferred-date` option a date can be set to specify the expiration of a prefix. The `valid` and `preferred lifetimes` are counted down in real time. When the expiration date is reached, the prefix will no longer be advertised.

If not otherwise specified all prefixes configured on interfaces that originate IPv6 router advertisements are advertised with a valid lifetime of 2592000 seconds (30 days) and a preferred lifetime of 604800 seconds (7 days).

When **onlink** is "on" (by default), the specified prefix is assigned to the link. Nodes sending traffic to such addresses that contain the specified prefix consider the destination to be locally reachable on the link.

When **autoconfig** is "on" (by default), it indicates to hosts on the local link that the specified prefix can be used for IPv6 autoconfiguration.

2.1.3 Timing Router Advertisements and Lifetimes

To configure the interval between IPv6 router advertisement transmissions on an interface, use the **ipv6 nd ra-interval** command in interface configuration mode.

```
Router (config-if)# ipv6 nd ra-interval seconds
```

The interval between transmissions defaults to 200 seconds. It should be less than or equal to the IPv6 router advertisement lifetime if the router is configured as a default router by using the **ipv6 nd ra-lifetime** command. To prevent synchronization with other IPv6 nodes, randomly adjust the actual value used to within 20 percent of the specified value.

To configure the "router lifetime" value in IPv6 router advertisements on an interface, use the **ipv6 nd ra-lifetime** command in interface configuration mode.

```
Router (config-if)# ipv6 nd ra-lifetime seconds
```

The "router lifetime" value defaults to 1800 seconds and is included in all IPv6 router advertisements sent out the interface. The value indicates the usefulness of the router as a default router on this interface. Setting the value to 0 indicates that the router should not be perceived a default router on this interface. The "router lifetime" value can be set to a non zero value to indicate that it should be considered a default router on this interface. The non zero value for the "router lifetime" value should not be less than the router advertisement interval.

To configure the amount of time that a remote IPv6 node is considered reachable, after some reachability confirmation event has occurred, use the **ipv6 nd reachable-time** command in interface configuration mode.

```
Router (config-if)# ipv6 nd reachable-time milliseconds
```

As a default 0 milliseconds (unspecified) is advertised in router advertisements and the value 30000 (30 seconds) is used for the neighbour discovery activity of the router itself.

The configured time enables the router to detect unavailable neighbors. Shorter configured times enable the router to detect unavailable neighbors more quickly; however, shorter times consume more IPv6 network bandwidth and processing resources in all IPv6 network devices. Very short configured times are not recommended in normal IPv6 operation.

The configured time is included in all router advertisements sent out of an interface so that nodes on the same link use the same time value. A value of 0 means indicates that the configured time is unspecified by this router.

2.1.3.1 Examples

a) The following example includes the IPv6 prefix `2001:0DB8::/64` in router advertisements sent out Ethernet `interface 0/0` with a valid lifetime of `1000` seconds, a preferred lifetime of `900` seconds, and both the "`onlink`" and "`autoconfig`" flags set:

```
Router(config)# interface ethernet 0/0
Router(config-if)# ipv6 nd prefix 2001:0DB8::/64 1000 \\
      900 onlink autoconfig
```

b) The following example configures an IPv6 router advertisement interval of `201` seconds for Ethernet `interface 0/0`:

```
Router(config)# interface ethernet 0/0
Router(config-if)# ipv6 nd ra-interval 201
```

c) The following example configures an IPv6 router advertisement lifetime of 1801 seconds for Ethernet `interface 0/0`:

```
Router(config)# interface ethernet 0/0
Router(config-if)# ipv6 nd ra-lifetime 1801
```

d) The following example configures an IPv6 reachable time of 1,700,000 milliseconds for Ethernet `interface 0/0`:

```
Router(config)# interface ethernet 0/0
Router(config-if)# ipv6 nd reachable-time 1700000
```

2.2 Juniper JunOS

To configure neighbour discovery on a JunOS platform, you include the following statements. Router advertisement are configured on a per-interface basis.

```
protocols {
    router-advertisement {
        interface interface-name {
            current-hop-limit number;
            default-lifetime seconds;
            (managed-configuration | no-managed-configuration);
            max-advertisement-interval seconds;
            min-advertisement-interval seconds;
            (other-stateful-configuration | no-other-stateful-
configuration);
            prefix prefix {
```

```
                (autonomous | no-autonomous);
                (on-link | no-on-link);
                preferred-lifetime seconds;
                valid-lifetime seconds;
            }
            reachable-time milliseconds;
            retransmit-timer milliseconds;
            traceoptions {
                file name <replace> <size size> <files number> <no-stamp>
                    <(world-readable | no-world-readable)>;
                flag flag <detail> <disable>;
            }
        }
    }
}
```

2.2.1 Configuring Prefix Information in Router Advertisements

You can set two fields in the router advertisement message to enable stateful autoconfiguration on a host: the `managed-configuration` field and the `other-stateful-configuration` field. Setting the managed configuration field enables the host to use a stateful autoconfiguration protocol for address autoconfiguration, along with any stateless autoconfiguration already configured. Setting the other stateful configuration field enables autoconfiguration of other nonaddress-related information.

By default, stateful autoconfiguration is disabled.

2.2.1.1 "on-link/not_on_link"

Router advertisement messages carry prefixes and information about them. A prefix is `onlink` when it is assigned to an interface on a specified link. The prefixes specify whether they are **`onlink`** or **`not-on-link`**. A node considers a prefix to be **`onlink`** if it is represented by one of the link's prefixes, a neighboring router specifies the address as the target of a redirect message, a neighbor advertisement message is received for the (target) address, or any neighbor discovery message is received from the address. These prefixes are also used for address autoconfiguration. The information about the prefixes specifies the lifetime of the prefixes, whether the prefix is autonomous, and whether the prefix is **`onlink`**.

You can specify prefixes in the router advertisement messages as **`onlink`**. When set as **`onlink`**, the prefixes are used for **`onlink`** determination. By default, prefixes are **`onlink`**.

To explicitly set prefixes as **`onlink`** or not **`onlink`**, include the **`on-link`** or **`not-on-link`** statement:

```
[edit protocols router-advertisement interface interface-name prefix prefix]
on-link | not-on-link;
```

2.2.1.2 "autonomous/no-autonomous"

You can specify prefixes in the router advertisement messages as autonomous. When set as autonomous, the prefixes are used for stateless address autoconfiguration. By default, prefixes are autonomous.

To explicitly specify prefixes as autonomous, include the **autonomous** or **no-autonomous** statement:

```
[edit protocols router-advertisement interface interface-name prefix prefix]
autonomous | no-autonomous;
```

2.2.1.3 "preferred-lifetime"

The preferred lifetime for the prefixes in the router advertisement messages specifies how long the prefix generated by stateless autoconfiguration remains preferred. By default, the preferred lifetime is set to 604,800 seconds.

To configure the preferred lifetime, include the preferred-lifetime statement:

```
[edit protocols router-advertisement interface interface-name prefix prefix]
preferred-lifetime seconds;
```

If you set the preferred lifetime to 0xffffffff, the lifetime is infinite.

The preferred lifetime value must never exceed the valid lifetime value.

2.2.1.4 "valid_lifetime"

The valid lifetime for the prefixes in the router advertisement messages specifies how long the prefix remains valid for **onlink** determination. By default, the valid lifetime is set to 2,592,000 seconds.

To configure the valid lifetime, include the valid-lifetime statement:

```
[edit protocols router-advertisement interface interface-name prefix prefix]
valid-lifetime seconds;
```

If you set the valid lifetime to `0xffffffff`, the lifetime is infinite.

The valid lifetime value must never be smaller than the preferred lifetime value.

2.2.1.5 "traceoptions"

To trace router advertisement traffic, you can specify options in the global traceoptions statement at the `[edit routing-options]` hierarchy level, and you can specify router advertisement options by including the **traceoptions** statement:

```
[edit protocols router-advertisement]
traceoptions {
    file name <replace> <size size> <files number> <no-stamp>
```

```
            <(world-readable | no-world-readable)>;
        flag flag <flag-modifier> <disable>;
    }
```

Note: Use the traceoption flags detail with caution. These flags may cause the CPU to become very busy.

2.2.2 Timing and Configuring Router Advertisements

2.2.2.1 "default-lifetime"

The default lifetime in router advertisement messages indicates the lifetime associated with the default router. To modify the default lifetime timer, include the default-lifetime statement:

```
[edit protocols router-advertisement interface interface-name]
default-lifetime seconds;
```

By default, the default router lifetime is three times the maximum advertisement interval.

2.2.2.2 "min/max-advertisement-interval"

The router sends router advertisements on each interface configured to transmit messages. The advertisements include route information and indicate to network hosts that the router is operational. The router sends these messages periodically, with a time range defined by minimum and maximum values.

To modify the router advertisement interval, include the **min-advertisement-interval** and **max-advertisement-interval** statements:

```
[edit protocols router-advertisement interface interface-name]
min-advertisement-interval seconds;
max-advertisement-interval seconds;
```

By default, the maximum advertisement interval is 600 seconds and the minimum advertisement interval is one-third the maximum interval, or 200 seconds.

2.2.2.3 "reachable-time"

After receiving a reachability confirmation from a neighbor, a node considers that neighbor reachable for a certain amount of time without receiving another confirmation. This mechanism is used for neighbor unreachability detection, a mechanism for finding link failures to a target node.

To modify the reachable time limit, include the reachable-time statement:

```
[edit protocols router-advertisement interface interface-name]
reachable-time milliseconds;
```

By default, the reachable time period is 0 milliseconds.

2.2.2.4 "current-hop-limit"

The **`current-hop-limit`** field in the router advertisement messages indicates the default value placed in the hop count field of the IP header for outgoing packets. To configure the hop limit, include the **`current-hop-limit`** statement:

```
[edit protocols router-advertisement interface interface-name]
current-hop-limit number;
```

The default hop limit is 64.

2.3 Quagga

Quagga is an advanced routing software package that provides TCP/IP based routing protocols. Information on Quagga in this document is based on the Quagga Manual for version 0.96. Quagga is started as a fork of GNU Zebra and incorporates the Zebra protocol today as part of the suite.

Currently Quagga supports GNU/Linux, BSD and Solaris. Below is a list of OS versions on which Quagga runs. Porting Quagga to other platforms is not so difficult. Platform dependent codes exist only in the *ZEBRA* daemon. Protocol daemons are platform independent.

- GNU/Linux 2.0.37
- GNU/Linux 2.2.x and higher
- FreeBSD 2.2.8
- FreeBSD 3.x
- FreeBSD 4.x
- NetBSD 1.4
- OpenBSD 2.5
- Solaris 2.6
- Solaris 7

Some IPv6 stacks are in development. Quagga supports the following IPv6 stacks. For BSD, we recommend KAME IPv6 stack. Solaris IPv6 stack is not yet supported.

- Linux IPv6 stack for GNU/Linux 2.2.x and higher.
- KAME IPv6 stack for BSD.
- INRIA IPv6 stack for BSD.

Since Quagga supports IPv6 in many ways, is easy and cheap to deploy, it is an ideal solution for IPv6 integration into networks where hardware routing devices are not yet IPv6 enabled or should not yet be involved in IPv6 routing for fear of corrupting IPv4 service. It is therefore expected that it might be used by network administrators who don't have any experience with it yet. For this reason

we will include rather a little more basic installation and configuration information about Quagga than we do for any other routing platform.

2.3.1 Installation

Quagga is installed like any other UNIX software by “configure”, “make” and “make install”. If no special options are specified with the configure script, all deamons as well as IPv6 support will be compiled into the binaries.

Quagga daemons have their own terminal interface or VTY. After installation, you have to setup each daemon'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
```

2.3.2 Basic Configuration

Each of the daemons has its own config file, which are usually located in “`/usr/local/etc`”. They are called `*.conf`, so for example the zebra configuration file is “`zebra.conf`”. However, you can specify any config file using the `-f` or `--config-file` options when starting the daemon.

Quagga can either be configured by editing the individual configuration files manually or via a command line interface very similar to that of Cisco IOS. The commands/directives are the same.

```
!
! 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.

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

```
password zebra!password
```

To configure Quagga/Zebra via the command line you can connect to the daemon using 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 zebra (version 0.96)
Copyright 1997-2000 Kunihiro Ishiguro


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
```

2.3.2.1 Basic Configuration Commands

Command Syntax:

```
Router(config)# hostname <hostname> {}
```

Set hostname of the router.

Command Syntax:

```
Router(config)# password <password> {}
```

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

Command Syntax:

```
Router(config)# enable password <password> {}
```

Set enable password.

Command Syntax:

```
Router(config)# (no) log stdout {}
```

Set/Unset logging output to **stdout**.

Command Syntax:

```
Router(config)# log file <filename> {}
```

If you want to log into a file please specify `filename` as follows:

```
Router(config)# log file /usr/local/etc/bgpd.log
```

Command Syntax:

```
Router(config)# (no) log syslog {}
```

Set/Unset logging output to syslog.

Command Syntax:

```
Router# write terminal {}
```

Displays the current configuration to the VTY interface.

Command Syntax:

```
Router(config)#  write file {}
```

Write current configuration to configuration file.

Command Syntax:

```
Router# configure terminal {}
```

Change to configuration mode. This command is the first step to configuration.

Conmand Syntax:

```
Router# terminal length <0-512> {}
```

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

Command Syntax:

```
Router#  who {}
Router# list {}
```

List commands.

Command Syntax:

```
Router(config)# service password-encryption {}
```

Encrypt password.

Command syntax:

```
Router(config)# service advanced-vty {}
```

Enable advanced mode VTY.

Command Syntax:

```
Router(config)# service terminal-length <0-512> {}
```

Set system wide line configuration. This configuration command applies to all VTY interfaces.

Command Syntax:

```
Router# show version {}
```

Show the current version of the Quagga and its build host information.

Command Syntax:

```
Router(config)# line vty {}
```

Enter VTY configuration mode.

Command Syntax:

```
Router(config)# (no) banner motd default {}
```

Set default **motd** string. To delete a set banner string use the **no** keyword without any string specification.

Command Syntax:

```
exec-timeout <minute> {}
exec-timeout <minute> <second> {}
```

Set VTY connection timeout value. When only one argument is specified it is used for timeout value in minutes. Optional second argument is used for timeout value in seconds. Default timeout value is 10 minutes. When timeout value is 0, it means no timeout.

Command Syntax:

```
no exec-timeout {}
```

Do not perform timeout at all. This command is as same as `exec-timeout 0 0`.

Command Syntax:

```
access-class <access-list> {}
```

Restrict VTY connections with an access list.

2.3.3 Running Quagga/Zebra

The Quagga daemons can be started from "`/etc/init.d/<daemon>`" with several different options, which are the same for all daemons.

- **`-d'** or **`--daemon'**: Runs in daemon mode.
- **`-f file'** or **`--config_file=file'**: Set configuration file name.
- **`-h'** or **`--help'**: Display help.
- **`-i file'** or **`--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 different points in the network.
- **`-A address'** or **`--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'** or **`--vty_port=port'**: Set the VTY TCP port number. If set to 0 then the TCP VTY sockets will not be opened.
- **`-u user'** or **`--vty_addr=user'**: Set the user and group to run as.
- **`-v'** or **`--version'**: Print program version.

Besides the common invocation options, the Zebra itself has some specific options:

- **`-b'** or **`--batch'**: Runs in batch mode. Zebra parses configuration file and terminates immediately.
- **`-k'** or **`--keep_kernel'**: When Zebra starts up, don't delete old self inserted routes.
- **`-l'** or **`--log_mode'**: Set verbose logging on.
- **`-r'** or **`--retain'**: When program terminates, retain routes added by Zebra.

2.3.4 Configuring Router Advertisements and Neighbor Discovery

Router Advertisements are configured per interface, so the following commands have to be entered in the context of an interface:

Command Syntax:

```
Router(config-if)# (no) ipv6 nd suppress-ra {}
```

Either switches of or on (**no** keyword) sending router advertisements on that interface.

Command Syntax:

```
Router(config-if)# (no) ipv6 nd prefix <ipv6prefix> [valid-lifetime] \\
                                [preferred-lifetime] [off-link] [no-autconfig] {}
```

This command configures 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 which 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 advertisements 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.

Command Syntax:

```
Router(config-if)# (no) ipv6 nd ra-interval <seconds> {}
```

The maximum time allowed between sending unsolicited multicast router advertisements from the interface, in seconds. Must be no less than 3 seconds. The default is 600. Omit `<seconds>` to delete the statement with the **`no`** keyword.

Command Syntax:

```
Router(config-if)# (no) ipv6 nd ra-lifetime <seconds> {}
```

This command sets the value to be placed in the Router Lifetime field of router advertisements sent from the interface, in seconds. It 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 the connected link. The value must be either zero or between the value specified with **`ipv6 nd ra-interval`** (or default) and 9000 seconds. The default is 1800 seconds. Omit `<seconds>` to delete the statement with the **`no`** keyword.

Command Syntax:

```
Router(config-if)# (no) ipv6 nd reachable-time <milliseconds> {}
```

Use this command to set the value to be placed in the Reachable Time field in the router advertisements messages sent by the router, in milliseconds. The configured time enables the router to detect unavailable neighbors. If set to zero it means this time is unspecified (by this router). The value set must be no greater than `3,600,000` milliseconds (1 hour). The default is 0. Omit `<milliseconds>` to delete the statement with the **`no`** keyword.

Command Syntax:

```
Router(config-if)# (no) ipv6 nd managed-config-flag {}
```

This statement sets/unsets the 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. As a default this flag is not set.

Command Syntax:

```
Router(config-if)# (no) ipv6 nd other-config-flag {}
```

This statement sets/unsets the flag in IPv6 router advertisements which indicates to hosts that they should use administered (stateful) protocol to obtain autoconfiguration information other than addresses. As a default this flag is not set.

A sample IPv6 interface may be configured as follows:

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

This interface will send out router advertisements for the prefix `2001:0DB8:5009::/64` every 600 seconds.

2.4 "Radvd"

"Radvd" stands for "Router Advertisement Daemon". It can be installed on any Linux system, which acts as a default IPv6 gateway router.

Radvd is configured in a configuration file that is typically located at `/etc` and called `radvd.conf`. It includes information about the prefix(es) to be advertised, lifetimes and (optionally) the frequency of router advertisements.

Following is a typical Radvd configuration file:

```
interface eth0 {
        AdvSendAdvert on;
        MinRtrAdvInterval 3;
        MaxRtrAdvInterval 10;
        Prefix 2001:638:500:101::/64 {
                AdvOnLink on;
                AdvAutonomous on;
                AdvRouterAddr on;
        };
};
```

For more information about configuration options and runtime parameters please refer to the manpage radvd(8).

3 Implementing Static Routing for IPv6

3.1 Cisco IOS

3.1.1 Prerequisites

The minimum required IOS release for the static routing feature in each train is generally the one where IPv6 became first available, that is 12.0(21)ST, 12.0(22)S, 12.2(2)T, 12.2(14)S, 12.3(2)T, 12.3(1)M.

Before configuring the router with a static IPv6 route you must enable the forwarding of IPv6 packets using the **`ipv6 unicast-routing`** global configuration command. Then enable IPv6 on at least one interface by configuring an IPv6 address on it, which can be accomplished with the command **`ipv6 address`** in the respective interface's context.

3.1.2 Configuring Static Routes

Using static routes in IPv6 is similar to configuring static routes for IPv4 with a few differences. A new command, `ipv6 route`, is used to configure IPv6 static routes and some of the IPv4 keywords are not yet supported. The command structure is as follows:

```
Router(config)# ipv6 route ipv6-prefix/prefix-length \\
      {ipv6-address | interface-type interface-number \\
      [ipv6-address]} [administrative-distance]
```

Note that a floating static route must be configured with a greater administrative distance parameter than any dynamic routing protocol because routes with smaller administrative distances are preferred. By default, static routes have smaller administrative distances than dynamic routes.

3.1.2.1 Examples

a) In the following example the IPv6 default route is configured to point to serial interface 2/0:
```
Router(config) # ipv6 route ::/0 serial 2/0
```
b) This example shows a normal route configured to go through serial interface 0:
```
Router(config) # ipv6 route 2001:0db8::/32 serial 0
```
c) In this example an additional (optional) IPv6 address is configured for the next hop:
```
Router(config) # ipv6 route 2001:0db8::/32 ethernet 0 fe80::1
```
d) The following example shows how a static route can be configured with just the next hop IPv6 address omitting any interface specification:
```
Router(config) # ipv6 route 2001:0db8::/32 2002:806b:f0fe::1
```

3.1.3 Verifying Static IPv6 Route Configuration and Operation

To verify that static routes have been correctly configured one can use the **`show ipv6 route`** command, which has the following syntax:

```
router # show ipv6 route [ipv6-address | \\
      ipv6-prefix/prefix-length | protocol]
```

3.1.3.1 Examples

a) In the following example the **show ipv6 route** EXEC command is used to verify the configuration of a static route through a point-to-point interface:

```
Router# show ipv6 route
IPv6 Routing Table - 9 entries
Codes: C - Connected, L - Local, S - Static, R - RIP, B - BGP
U - Per-user Static route
I1 - ISIS L1, I2 - ISIS L2, IA - ISIS interarea
O - OSPF intra, OI - OSPF inter, OE1 - OSPF ext 1, OE2 - OSPF ext 2
S 2001:0DB8::/32 [1/0]
via ::, Serial2/0
```

b) In this case, the **show ipv6 route** EXEC command is used to verify the configuration of a static route on a multi-access interface. An IPv6 link-local address (FE80::1) is the next hop router.

```
Router# show ipv6 route
IPv6 Routing Table - 11 entries
Codes: C - Connected, L - Local, S - Static, R - RIP, B - BGP
U - Per-user Static route
I1 - ISIS L1, I2 - ISIS L2, IA - ISIS interarea
O - OSPF intra, OI - OSPF inter, OE1 - OSPF ext 1, OE2 - OSPF ext 2
S 2001:0DB8::/32 [1/0]
via FE80::1, Ethernet0/0
```

c) In this example, the **show ipv6 route** EXEC command is used with **static** as the value of the protocol argument to display information about static routes installed in the IPv6 routing table:

```
Router# show ipv6 route static
IPv6 Routing Table - 330 entries
Codes: C - Connected, L - Local, S - Static, R - RIP, B - BGP
U - Per-user Static route
I1 - ISIS L1, I2 - ISIS L2, IA - ISIS interarea
S 2001:0DB8::/32 [1/0]
via ::, Tunnel0
S 3FFE:C00:8011::/48 [1/0]
via ::, Null0
```

```
S ::/0 [254/0]
via 2002:806B:F0FE::1, Null
```

d) In the following example, the **`debug ipv6 routing`** privileged EXEC command is used to verify the installation of a floating static route into the IPv6 routing table when an IPv6 RIP route is deleted. The floating static IPv6 route was previously configured with an administrative distance value of 130. The backup route was added as a floating static route because Routing Information Protocol (RIP) routes have a default administrative distance of 120 and the RIP route should be the preferred route. When the RIP route is deleted the floating static route is installed in the IPv6 routing table.

```
Router# debug ipv6 routing
*Oct 10 18:28:00.847: IPv6RT0: rip two, Delete 2001:0DB8::/32 from
table
*Oct 10 18:28:00.847: IPv6RT0: static, Backup call for 2001:0DB8::/32
*Oct 10 18:28:00.847: IPv6RT0: static, Add 2001:0DB8::/32 to table
*Oct 10 18:28:00.847: IPv6RT0: static, Adding next-hop :: over
Serial2/0 for 2001:0DB8::/32, [130/0]
```

3.2 Juniper JunOS

The only difference in specifiying static routes for IPv6 and IPv4 on Juniper platforms is that for the new version of the Internet Protocol one needs to explicitly specify the IPv6 routing table (`inet6.0`, `inet6.2`) in the statements. Other than that, syntax and options are very much the same and therefore are not explained in this section. Please refer to Section 3.2.1.1 for examples.

3.2.1.1 Examples

a) Configure an IPv6 default route through the next-hop router `2001:638:500::1`:

```
[edit]
user@host# set routing-options rib inet6.0 static \\
      route abcd::/48 next-hop 2001:638:500::1


[edit]
user@host# show
routing-options {
    static {
        route abcd::/48 next-hop 2001:638:500::1;
    }
}
```

b) Install an IPv6 static route into both `inet6.0` and `inet6.2` routing tables:

```
[edit routing-options rib table1.inet6.0 static]
```

```
rib-group groupA;

[edit routing-options rib-groups]
groupA {
    import-rib [table1.inet6.0 inet6.0 inet6.2]
}
```

c) Propagate IPv6 static routes into the routing protocols:

```
[edit routing-options rib inet6.0 static (defaults | route)]
discard;
```

d) Resolve an IPv6 static route to non-next-hop router `2001:638:500:101::1/64` using next-hop router `2001:638:500::1`:

```
[edit]
user@host# set routing-options rib inet6.0 static route
2001:638:500:101::1/64 next-hop 2001:638:500::1 resolve

[edit]
user@host# show route 2001:638:500:101::/64
inet6.0: 26 destinations, 27 routes (25 active, 0 holddown, 1 hidden)
+ = Active Route, - = Last Active, * = Both

2001:638:500:101::/64             *[Static/5] 00:01:50
                    > to 2001:638:500::2 via ge-0/1/0.0

user@host# show route 2000::1
inet6.0: 26 destinations, 27 routes (25 active, 0 holddown, 1 hidden)
+ = Active Route, - = Last Active, * = Both

2000::/126      *[BGP/170] 00:05:32, MED 20, localpref 100
                AS path: 2 I
                > to 2001:638:500::2 via ge-0/1/0.0
```

3.3 Quagga/Zebra

Command Syntax:

```
Router(config)# ipv6 route <network> <gateway> [<distance>] {}
```

`network` is the destination prefix. `gateway` indicates the gateway for that prefix. As `gateway` you can either specify an IPv6 address or interface name. If the interface name is `null0` then Zebra installs a blackhole route. `distance` is an optional parameter to install the route with a specified distance.

Command Syntax:

```
Router(config)# table <tableno> {}
```

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.

Command Syntax:

```
Router# show ipv6 route [<route>]{}
```

This command can be used to display routes in the current selected routing table. To show a specific route, specify it at the end.

4 Implementing RIPng for IPv6

IPv6 RIP or RIPng (RIP Next Generation) is generally very similar to RIP (for IPv4) and functions in the same way, offering the same benefits. RIP enhancements for IPv6 (RIPng) is detailed in RFC 2080 and includes support for IPv6 addresses and prefixes, and the use of all-RIP-routers multicast group address FF02::9 as the destination address for RIP update messages.

The RIPng IGP uses the Bellman-Ford distance-vector algorithm to determine the best route to a destination. RIPng uses the hop count as the metric and allows hosts and routers to exchange information for computing routes through an IP-based network. RIPng is intended to act as an IGP for moderately-sized autonomous systems (ASs).

RIPng is a User Datagram Protocol (UDP)-based protocol and uses UDP port 521.

RIPng has the following architectural limitations:

- The longest network path cannot exceed 15 hops (assuming that each network, or hop, has a cost of 1).
- RIPng depends on counting to infinity to resolve certain unusual situations. When the network consists of several hundred routers, and when a routing loop has formed, the amount of time and network bandwidth required to resolve a next hop might be great.
- RIPng uses only a fixed metric to select a route. Other IGPs use additional parameters, such as measured delay, reliability and load.

A RIPng packet header contains the following fields:

- Command—Indicates whether the packet is a request or response message. Request messages seeks information for the router's routing table. Response messages are sent periodically or when a request message is received. Periodic response messages are called update messages. Update messages contain the command and version fields and a set of destinations and metrics.
- Version number—Specifies the version of RIPng that the originating router is running. This is currently set to Version 1.

The rest of the RIPng packet contains a list of routing table entries with the following fields:

- Destination prefix—128-bit IPv6 address prefix for the destination.
- Prefix length—Number of significant bits in the prefix.
- Metric—Value of the metric advertised for the address.
- Route tag—A route attribute that must be advertised and redistributed with the route. Primarily, the route tag distinguishes external RIPng routes from internal RIPng routes in cases where routes must be redistributed across an exterior gateway protocol (EGP).

4.1 Cisco IOS

In the Cisco IOS software implementation of IPv6 RIP each IPv6 RIP process maintains a local routing table, referred to as a Routing Information Database (RIB). The IPv6 RIP RIB contains a set of best-cost IPv6 RIP routes learned from all its neighbouring networking devices. If IPv6 RIP learns

the same route from two different neighbours, but with different costs, it will store only the lowest cost route in the local RIB. The RIB also stores any expired routes that the RIP process is advertising to its neighbours running RIP. IPv6 RIP will try to insert every non-expired route from its local RIB into the master IPv6 RIB. If the same route has been learned from a different routing protocol with a better administrative distance than IPv6 RIP, the RIP route will not be added to the IPv6 RIB but the RIP route will still exist in the IPv6 RIP RIB.

4.1.1 Prerequisites

For the minimum required IOS version for IPv6 RIP features please refer to:

http://www.cisco.com/univercd/cc/td/doc/product/software/ios123/123cgcr/ipv6_c/ftipv6s.htm#wp10 04964

As with all IPv6 features within IOS the same configuration prerequisites apply to configure IPv6 RIP features. Therefore IPv6 forwarding must be enabled and an IPv6 address must be configured for at least one interface, in this case specifically those which take part in RIP routing. See section 3.1.1 about prerequisites for the configuration of static routes for more details.

4.1.2 How to Implement RIP for IPv6

When configuring supported routing protocols in IPv6, you must create the routing process, enable the routing process on interfaces and customize the routing protocol for your particular network.

Note: The following sections describe the configuration tasks for creating an IPv6 RIP routing process and enabling the routing process on interfaces. The following sections do not provide in-depth information on customizing RIP because the protocol functions the same in IPv6 as it does in IPv4.

To configure IPv6 on a Cisco IOS platform one has to perform two tasks. The first is to create an IPv6 RIP routing process with a specific `name`. The syntax of the corresponding command is:

```
router(config)# ipv6 router rip name
```

This command enters the configuration context for this routing process to optionally configure it further. In its simplest form one has to do nothing here but exit the context again.

The second task in configuring IPv6 RIP is to enable the process on the interfaces involved in RIP routing. In order to accomplish this it's necessary to enter the corresponding interface's context and use the following command.

```
router(config-if)# ipv6 rip name enable
```

By "`name`" one refers to the respective IPv6 RIP routing process.

4.1.2.1 Examples

a) The following example command creates an IPv6 RIP routing process by the name of "`ciscotest`"

```
Router(config)# ipv6 router rip ciscotest
```

b) The routing process configured in example a) is now enabled on ethernet interface 0/0:

```
Router(config)# interface Ethernet 0/0
Router(config-if)# ipv6 rip ciscotest enable
```

4.1.3 Customizing IPv6 RIP

This optional task explains how to configure the maximum numbers of equal-cost paths that IPv6 RIP will support, adjust the IPv6 RIP timers and originate a default IPv6 route.

The corresponding commands are either entered in the configuration context of IPv6 RIP routing processes or applied to specific interfaces in their configuration contexts.

The optional command for defining the maximum number of equal-cost routes that IPv6 RIP can support applies to the whole routing process:

```
Router(config-router)# maximum-paths number-paths
```

The `number-paths` argument is an integer from 1 to 4. The default for RIP is 4 paths.

RIP update times are also relevant for the routing process in general. By default, updates occur every 30 seconds and timeout after 180 seconds, hold-down lasts 0 seconds and garbage collection is activated after 120 seconds. To change these values use the following command:

```
Router(config-router)# timers update timeout holddown \\
                                                garbage-collection
```

To originate the IPv6 default route (::/0) into the updates of a specified RIP routing process from a specified interface enter the following command in the configuration context of that interface:

```
Router(config-if)# ipv6 rip name default-information \\
                                                {only | originate}
```

Note: To avoid routing loops after the IPv6 default route (::/0) is originated out of any interface, the routing process ignores all default routes received on any interface.

Specifying the **`only`** keyword originates the default route (::/0) but suppresses all other routes in the updates sent on this interface.

Specifying the **`originate`** keyword originates the default route (::/0) in addition to all other routes in the updates sent on this interface.

4.1.3.1 Examples

a) This example shows how the RIP process is configured to support only one equal cost path:

```
Router(config-router)# maximum-paths 1
```

b) With the following command the update timer is reduced to 5 seconds, timeouts occur after 15 seconds while holddown lasts 10 seconds and garbage collection is activated after 30 seconds:

```
Router(config-router)# timers 5 15 10 30
```

c) In this example Ethernet interface 0/0 is configured to originate the IPv6 default route in addition to all other routes for IPv6 RIP process "`ciscotest`":

```
Router(config)# interface Ethernet 0/0
Router(config-if)# ipv6 rip ciscotest \\
                                        default-information originate
```

4.1.4 IPv6 prefix lists to filter routes on outgoing or incoming RIP updates

IPv6 prefix lists are used to specify certain prefixes or a range of prefixes that must be matched before a permit or deny statement can be applied. Two operand keywords can be used to designate a range of prefix lengths to be matched. A prefix length of less than, or equal to, a value is configured with the **le** keyword. A prefix length greater than, or equal to, a value is specified using the **ge** keyword. The **ge** and **le** keywords can be used to specify the range of the prefix length to be matched in more detail than the usual `ipv6-prefix/prefix-length` argument. For a candidate prefix to match against a prefix list entry three conditions can exist:

- the candidate prefix must match the specified prefix list and prefix length entry;
- the value of the optional **le** keyword specifies the range of allowed prefix lengths from the `prefix-length` argument up to, and including, the value of the **le** keyword;
- the value of the optional **ge** keyword specifies the range of allowed prefix lengths from the value of the **ge** keyword up to, and including, 128.

Note: The first condition must match before the other conditions take effect.

An exact match is assumed when the **ge** or **le** keywords are not specified. If only one keyword operand is specified then the condition for that keyword is applied, and the other condition is not applied. The `prefix-length` value must be less than the **ge** value. The **ge** value must be less than, or equal to, the **le** value. The **le** value must be less than or equal to 128.

An IPv6 prefix list is created by specifying its first entry in the global configuration environment. The command syntax to either deny specific routes or permit them is as follows:

Command syntax:

```
Router(config)# ipv6 prefix-list prefix-list-name \\
      [seq seq-number] {deny ipv6-prefix/prefix-length \\
                        | description text} [ge ge-value] [le le-value]


Router(config)# ipv6 prefix-list prefix-list-name \\
      [seq seq-number] {permit ipv6-prefix/prefix-length \\
                        | description text} [ge ge-value] [le le-value]
```

The optional **seq** `seq-number` parameter specifies the sequence number of the corresponding entry in the specified `prefix-list`.

To enter more than one filter to any prefix list repeat the above commands specifying the same `prefix-list-name` as often as needed with the appropriate parameters.

IPv6 prefix lists can be applied to specific IPv6 RIP routing updates that are received or sent on the specific interface with the following command:

Command syntax:

```
Router(config-router)# distribute-list prefix-list \\
      prefix-list-name {in | out} \\
      [interface-type interface-number]
```

Note: If an interface is not specified with the **`distribute-list prefix-list`** command, the specified prefix list is applied to IPv6 routing updates that are sent or received on all interfaces in the router running the specified process.

The **`in`** keyword applies the prefix list to incoming routing updates on the specified interface.

The **`out`** keyword applies the prefix list to outgoing routing updates on the specified interface.

4.1.4.1 Examples

a) In this example, the default IPv6 route will be denied.

```
Router(config)# ipv6 prefix-list eth0/0-in-flt seq 10 \\
      deny ::/0
```

b) In this example, all routes are permitted. This entry is following the deny rule in Step a) in order to permit all other routes, except the IPv6 default route. Without this permit rule all prefixes would be blocked.

```
Router(config)# ipv6 prefix-list eth0/0-in-flt seq 20 \\
      permit ::/0 le 128
```

c) In the following example, the **`ipv6 prefix-list`** "`eth0/0-in-flt`" is distributed to IPv6 RIP routing process "`ciscotest`". It is specifically used to filter routes coming in through interface Ethernet 0/0. The result is that all routes but the IPv6 default route are accepted.

```
Router(config)# ipv6 router rip ciscotest
Router(config-router)# distribute-list prefix-list eth0/0-in-flt \\
      in Ethernet 0/0
```

4.1.5 Route Tags and Redistribution of Routes into an RIP Routing Process

This task explains how to set route tags using a route map and redistribute tagged routes into an IPv6 RIP routing process.

Defining route maps and tagging routes for IPv6 is not much different from the same task in IPv4. The only difference is the fact that route maps match to IPv6 prefix lists.

Command syntax:

```
Router(config)# route-map map-tag [permit | deny] \\
      [sequence-number]
Router(config-route-map)# match ipv6 address prefix-list \\
      prefix-list-name
Router(config-route-map)# set tag value
```

Note: IPv6 RIP does not support route tags when the number of parallel routes is configured to be greater than one. Use the **`maximum-paths`** command to configure the number of parallel routes.

Redistribution of routes defined in a route map is then set in the respective IPv6 RIP process with the **redistribute** command:

```
Router(config-router)# redistribute protocol [process-id] \\
      {level-1 | level-1-2 | level-2} [metric metric-value] \\
      [metric-type {internal | external}] [route-map map-name]
```

The protocol argument can be one of the following keywords: **bgp**, **connected**, **isis**, **rip**, or **static**.

The **rip** keyword and **process-id** argument specify an IPv6 RIP routing process.

Note: The **connected** keyword refers to routes that are established automatically by assigning IPv6 addresses to an interface.

4.1.5.1 Examples

This example shows how a route map with the name "bgp-to-rip" is defined to match a previously defined IPv6 prefix list with the name "bgp-to-rip-flt". This route map is then used to tag those routes with the tag "4" and to subsequently be used to redistribute them in an RIP routing process:

```
Router(config) # route-map bgp-to-rip permit 10
Router(config-route-map)# match ipv6 address prefix-list \\
      bgp-to-rip-flt
Router(config-route-map)# set tag 4
Router(config-route-map)# exit
Router(config)# ipv6 router rip ciscotest
Router(config-router)# redistribute bgp 65001 route-map \\
      bgp-to-rip
```

4.1.6 Verifying IPv6 RIP Configuration and Operation

This task explains how to display information to verify the configuration and operation of IPv6 RIP. The commands, which are specific to RIP in this context are **show ipv6 rip** and **debug ipv6 rip**, but as with static routes **show ipv6 route** is also useful in this task. To display information about current IPv6 RIP processes use the following command:

```
Router> show ipv6 rip [name] [database | next-hops]
```

Note that this command may be used as an unprivileged user.

Debugging messages to IPv6 RIP routing transactions may be displayed with the following command syntax:

```
Router# debug ipv6 rip [interface-type interface-number]
```

The debugging information may be constrained to just those transactions taking place on a specific interface. The command may only be executed by privileged users.

For the syntax of the **show ipv6 route** command please refer to section 3.1.3 in the static route chapter.

4.1.6.1 Examples

a) In the following example, output information about all current IPv6 RIP processes is displayed using the **`show ipv6 rip`** user EXEC command:

```
Router> show ipv6 rip
RIP process "cisco", port 521, multicast-group FF02::9, pid 62
Administrative distance is 120. Maximum paths is 1
Updates every 5 seconds, expire after 15
Holddown lasts 10 seconds, garbage collect after 30
Split horizon is on; poison reverse is off
Default routes are generated
Periodic updates 223, trigger updates 1
Interfaces:
Ethernet0/0
Redistribution:
Redistributing protocol bgp 65001 route-map bgp-to-rip
```

b) In the following example, output information about a specified IPv6 RIP process database is displayed using the **`show ipv6 rip`** user EXEC command with the *name* argument and the **`database`** keyword. In the following output for the IPv6 RIP process named `ciscotest`, timer information is displayed, and route `4002::16/64` has a route tag set:

```
Router> show ipv6 rip ciscotest database
RIP process "ciscotest", local RIB
2000::/64, metric 2
Ethernet0/0/FE80::A8BB:CCFF:FE00:B00, expires in 13 secs
4000::/16, metric 2 tag 4, installed
Ethernet0/0/FE80::A8BB:CCFF:FE00:B00, expires in 13 secs
4001::/16, metric 2 tag 4, installed
Ethernet0/0/FE80::A8BB:CCFF:FE00:B00, expires in 13 secs
4002::/16, metric 2 tag 4, installed
Ethernet0/0/FE80::A8BB:CCFF:FE00:B00, expires in 13 secs
::/0, metric 2, installed
Ethernet0/0/FE80::A8BB:CCFF:FE00:B00, expires in 13 secs
```

c) In the following example, output information for a specified IPv6 RIP process is displayed using the **`show ipv6 rip`** user EXEC command with the *name* argument and the **`next-hops`** keyword:

```
Router> show ipv6 rip ciscotest next-hops
RIP process "ciscotest", Next Hops
FE80::A8BB:CCFF:FE00:A00/Ethernet0/0 [4 paths]
```

d) In the following example, output information for all IPv6 RIP routes is displayed using the **show ipv6 route** user EXEC command with the **rip** protocol keyword:

```
Router> show ipv6 route rip
IPv6 Routing Table - 17 entries
Codes: C - Connected, L - Local, S - Static, R - RIP, B - BGP
U - Per-user Static route
I1 - ISIS L1, I2 - ISIS L2, IA - ISIS interarea
O - OSPF intra, OI - OSPF inter, OE1 - OSPF ext 1, OE2 - OSPF ext 2
R 2001:1::/32 [120/2]
via FE80::A8BB:CCFF:FE00:A00, Ethernet0/0
R 2001:2::/32 [120/2]
via FE80::A8BB:CCFF:FE00:A00, Ethernet0/0
R 2001:3::/32 [120/2]
via FE80::A8BB:CCFF:FE00:A00, Ethernet0/0
```

e) In the following example, debugging messages for IPv6 RIP routing transactions are displayed using the **debug ipv6 rip** privileged EXEC command:

```
Router# debug ipv6 rip
RIPng: Sending multicast update on Ethernet0/0 for ciscotest
src=FE80::A8BB:CCFF:FE00:B00
dst=FF02::9 (Ethernet0/0)
sport=521, dport=521, length=112
command=2, version=1, mbz=0, #rte=5
tag=0, metric=1, prefix=2000::/64
tag=4, metric=1, prefix=4000::/16
tag=4, metric=1, prefix=4001::/16
tag=4, metric=1, prefix=4002::/16
tag=0, metric=1, prefix=::/0
RIPng: Next RIB walk in 10032
RIPng: response received from FE80::A8BB:CCFF:FE00:A00 on Ethernet0/0
for ciscotest
```

```
src=FE80::A8BB:CCFF:FE00:A00 (Ethernet0/0)
dst=FF02::9
sport=521, dport=521, length=92
command=2, version=1, mbz=0, #rte=4
tag=0, metric=1, prefix=2000::/64
tag=0, metric=1, prefix=2000:1::/32
tag=0, metric=1, prefix=2000:2::/32
```

Note: By default, the system sends the output from debug commands and system error messages to the console. To redirect debugging output, use the logging command options within privileged EXEC mode. Possible destinations include the console, virtual terminals, internal buffer and UNIX hosts running a syslog server. For complete information on debug commands please refer to the appropriate Cisco IOS manual.

4.1.7 Complete Configuration Example for IPv6 RIP

In the following example, the IPv6 RIP process named `ciscotest` is enabled on the router and on Ethernet interface 0/0. The IPv6 default route (::/0) is advertised in addition to all other routes in router updates sent on Ethernet interface 0/0. Additionally, BGP routes are redistributed into the RIP process named `ciscotest` according to a route map where routes that match a prefix list are also tagged. The number of parallel paths is set to one to allow the route tagging, and the IPv6 RIP timers are adjusted. A prefix list named `eth0/0-in-flt` filters inbound routing updates on Ethernet interface 0/0.

```
ipv6 router rip ciscotest
 maximum-paths 1
 redistribute bgp 65001 route-map bgp-to-rip
 timers 5 15 10 30
 distribute-list prefix-list eth0/0-in-flt in Ethernet0/0
!
interface Ethernet0/0
 ipv6 address 2000::/64 eui-64
ipv6 rip ciscotest enable
ipv6 rip ciscotest default-information originate
!
ipv6 prefix-list bgp-to-rip-flt seq 10 deny 4003::/16 le 128
ipv6 prefix-list bgp-to-rip-flt seq 20 permit 4000::/8 le 128
!
ipv6 prefix-list eth0/0-in-flt seq 10 deny ::/0
ipv6 prefix-list eth0/0-in-flt seq 15 permit ::/0 le 128
!
```

```
route-map bgp-to-rip permit 10
 match ipv6 address prefix-list bgp-to-rip-flt
 set tag 4
```

4.2 Juniper JunOS

The JUNOS software implementation of RIPng is similar to RIPv2. However, RIPng is a distinct routing protocol from RIPv2 and differs in a few aspects from the older version of RIP. Aside from the general architecutral limitations of RIPng as stated at the beginning of this chapter the following additional differences are specifc to JunOS in comparison to RIPv2:

- RIPng does not need to implement authentication on packets.
- There is no support for multiple instances of RIPng.
- There is no support for RIPng routing table groups.

4.2.1 Configuring RIPng

To configure Routing Information Protocol Next-Generation (RIPng), you include the following statements:

```
protocols {
    ripng {
        graceful-restart {
            disable;
            restart-time seconds;
        }
        holddown seconds;
        import [ policy-names ];
        metric-in metric;
        receive <none>;
        send <none>;
        traceoptions {
            file name <replace> <size size> <files number> <no-stamp>
                <(world-readable | no-world-readable)>;
            flag flag <flag-modifier> <disable>;
        }
        group group-name {
            export [ policy-names ];
            metric-out metric;
            preference number;
            neighbor neighbor-name {
                import [ policy-names ];
```

```
                metric-in metric;
                receive <none>;
                send <none>;
            }
        }
    }
}
```

4.2.2 Define RIPng Global Properties

When RIPng detects a route with a high metric associated, the router waits for a period of time before making any updates into the routing table. This minimizes the effects of route flapping to the routing table. The period of time that RIPng waits is the hold-down timer.

To configure the hold-down timer for RIPng, include the **holddown** statement, which can be a value from 10 through 180. The default value is 180 seconds.

Graceful restart is disabled by default. You can globally enable graceful restart for all routing protocols under the [**edit routing-options**] hierarchy level.

You can configure graceful restart parameters specifically for RIPng. To do this, include the **graceful-restart** statement.

To disable graceful restart for RIPng, specify the **disable** statement. To configure a time period for the restart to finish, specify the **restart-time** statement.

4.2.3 Define RIPng Global and Neighbour-Specific Properties

To define RIPng properties can be entered either in the global RIPng configuration context and/or with each neighbour. They either apply to all or just to a specific RIPng neighbour.

```
[edit protocols ripng ]
import  [ policy-names ];
metric-in  metric ;
receive  receive-options ;
send  send-options ;

[edit protocols ripng group group-name]
neighbor neighbor-name {
    import [ policy-names ];
    metric-in metric;
    receive receive-options;
    send send-options;
}
```

By default, RIPng imports routes from the neighbours configured with the **neighbor** statement. These routes include those learned from RIPng as well as those learned from other protocols. By default, routes that RIPng imports from its neighbors have a metric of 1 added to the current route metric.

To change the default metric to be added to incoming routes, include the **metric-in** statement, which can be a value from 1 through 15. A value of 16 indicates infinity or unreachable.

You can enable and disable the sending or receiving of update messages. By default, sending and receiving update messages is enabled. To disable the sending and receiving of update messages, include the **receive** and **send** statements with the **none** keyword

To enable the sending and receiving of update messages, just include the **receive** and **send** statements.

To filter routes being imported by the local router from its neighbors, include the **import** statement and list the names of one or more policies to be evaluated. If you specify more than one policy, they are evaluated in order (first to last) and the first matching policy is applied to the route. If no match is found, the local router does not import any routes.

4.2.4 Defining RIPng Neighbour-Specific Properties

By default, RIPng does not export routes it has learned to its neighbors. To have RIPng export routes, apply one or more export policies. To apply export policies and to filter routes being exported from the local router to its neighbors, include the **export** statement and list the name of the policy to be evaluated.

You can define one or more export policies. If no routes match the policies, the local router does not export any routes to its neighbors. Export policies override any metric values determined through calculations involving the **metric-in** (see above) and **metric-out** values.

If you configure an export policy, RIPng exports routes it has learned to the neighbors configured with the **neighbor** statement.

If a route being exported was learned from a member of the same RIPng group, the metric associated with that route (unless modified by an export policy) is the normal RIPng metric. For example, a RIPng route with a metric of 5 learned from a neighbor configured with a **metric-in** value of 2 is advertised with a combined metric of 7 when advertised to RIPng neighbors in the same group. However, if this route was learned from a RIPng neighbor in a different group or from a different protocol, the route is advertised with the metric value configured for that group with the **metric-out** statement. The default value for **metric-out** is 1.

By default, the JunOS software assigns a preference of 100 to routes that originate from RIPng. When the JunOS software determines a route's preference to become the active route, the software selects the route with the lowest preference and installs this route into the forwarding table.

To modify the default RIPng preference value, include the **preference** statement, which can be a value from 0 through 4,294,967,295 (232 - 1).

4.2.5 Tracing RIPng traffic

To trace RIPng protocol traffic, you can specify options in the global **traceoptions** statement at the [**edit routing-options**] hierarchy level, and specify RIPng-specific options by including the traceoptions statement:

You can specify the following RIPng-specific options in the RIPng **traceoptions** statement:

- **all**—Trace everything.
- **error**—Trace RIPng errors.
- **expiration**—Trace RIPng route expiration processing.
- **general**—Trace general events.
- **holddown**—Trace RIPng hold-down processing.
- **normal**—Trace normal events.
- **packets**—Trace all RIPng packets.
- **policy**—Trace policy processing.
- **request**—Trace RIPng information packets.
- **route**—Trace routing information.
- **state**—Trace state transitions.
- **task**—Trace routing protocol task processing.
- **timer**—Trace routing protocol timer processing.
- **trigger**—Trace RIPng triggered updates.
- **update**—Trace RIPng update packets.

Note: Use the traceoption flags **detail** and **all** with caution. These flags may cause the CPU to become very busy.

4.2.6 Complete Configuration Example

Configure RIPng:

```
[edit policy-options]
policy-statement redist-direct {
    from protocol direct;
    then accept;
```

```
}
[edit protocols ripng]
metric-in 3;
group wan {
    metric-out 2;
    export redist-direct;
    neighbor so-0/0/0.0;
    neighbor at-1/1/0.0;
    neighbor at-1/1/0.42;
    neighbor at-1/1/1.42 {
        receive version-2;
    }
}
group local {
    neighbor ge-2/3/0.0 {
        metric-in 1;
        send broadcast;
    }
}
```

4.3 Quagga

There are no `ripngd` specific invocation options. Common options can be specified (see section 2.3.3).

Currently `ripngd` supports the following commands:

Command Syntax:

```
Router(config)# router ripng {}
```

This command is used to enable RIPng.

Command Syntax:

```
Router(config-ripng)# flush_timer time {}
```

Set flush timer.

Command Syntax:

```
Router(config-ripng)# network <network> {}
```

Set RIPng enabled interface by `network`.

Command Syntax:

```
Router(config-ripng)# network <ifname> {}
```

Set RIPng enabled interface by `ifname`.

Command Syntax:

```
Router(config-ripng)# route <network> {}
```

Set RIPng static routing announcement of `network`.

Command Syntax:

```
Router# router zebra {}
```

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

To verify configuration or to debug it, several `show` and `debug` commands can be used:

Commands:

```
Router# show ip ripng {}
Router# show debugging ripng {}
Router# show ripng events {}
Router# debug ripng packet {}
Router# debug ripng zebra {}
```

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.

```
Router(config-ripng)# distribute-list access_list (in|out) ifname {}
```

Example:

```
Router(config-ripng) # distribute-list local-only out sit1
```

5 Implementing IS-IS for IPv6

IS-IS in IPv6 functions the same and offers many of the same benefits as IS-IS in IPv4. IPv6 enhancements to IS-IS allow IS-IS to advertise IPv6 prefixes in addition to IPv4 and OSI routes.

5.1 Cisco IOS

Extensions to the IS-IS command-line interface (CLI) allow configuration of IPv6-specific parameters. IPv6 IS-IS extends the address families supported by IS-IS to include IPv6, in addition to OSI and IPv4.

IPv6 IS-IS in IOS supports either single-topology mode or multiple topology mode.

Single-topology support for IPv6 allows IS-IS for IPv6 to be configured on interfaces along with other network protocols (for example, IPv4 and Connectionless Network Service [CLNS]). All interfaces must be configured with the identical set of network address families. In addition, all routers in the IS-IS area (for Level 1 routing) or the domain (for Level 2 routing) must support the identical set of network layer address families on all interfaces.

When single-topology support for IPv6 is being used, either old or new-style TLVs may be used. However, the TLVs used to advertise reachability to IPv6 prefixes use extended metrics. Cisco routers do not allow an interface metric to be set to a value greater than 63 if the configuration is not set to support only new-style TLVs for IPv4. In single-topology IPv6 mode, the configured metric is always the same for both IPv4 and IPv6.

IS-IS multitopology support for IPv6 allows IS-IS to maintain a set of independent topologies within a single area or domain. This mode removes the restriction that all interfaces on which IS-IS is configured must support the identical set of network address families. It also removes the restriction that all routers in the IS-IS area (for Level 1 routing) or domain (for Level 2 routing) must support the identical set of network layer address families. Because multiple SPFs are performed, one for each configured topology, it is sufficient that connectivity exists among a subset of the routers in the area or domain for a given network address family to be routable.

You can use the **`isis ipv6 metric`** command to configure different metrics on an interface for IPv6 and IPv4.

When multitopology support for IPv6 is used, enter the **`metric-style wide`** command to configure IS-IS to use new-style TLVs because TLVs used to advertise IPv6 information in LSPs are defined to use only extended metrics.

Transition from Single-Topology to Multitopology Support for IPv6

All routers in the area or domain must use the same type of IPv6 support, either single-topology or multitopology. A router operating in multitopology mode will not recognize the ability of the single-topology mode router to support IPv6 traffic, which will lead to holes in the IPv6 topology. To transition from single-topology to a more flexible multitopology support, a multitopology transition mode is provided.

The multitopology transition mode allows a network operating in single-topology IS-IS IPv6 support mode to continue to work while upgrading routers to include multitopology IS-IS IPv6 support. While in transition mode, both types of TLVs (single-topology and multitopology) are sent in LSPs

for all configured IPv6 addresses, but the router continues to operate in single-topology mode (that is, the topological restrictions of the single-topology mode are still in effect). After all routers in the area or domain have been upgraded to support multitopology IPv6 and are operating in transition mode, transition mode can be removed from the configuration. Once all routers in the area or domain are operating in multitopology IPv6 mode, the topological restrictions of single-topology mode are no longer in effect.

5.1.1 Prerequisites

For the minimum required IOS version for IS-IS features please refer to:

http://www.cisco.com/univercd/cc/td/doc/product/software/ios123/123cgcr/ipv6_c/ftipv6s.htm#wp10 04964

Before configuring the router to run IPv6 IS-IS, globally enable IPv6 using the **`ipv6 unicast-routing`** global configuration command.

5.1.2 Restrictions for Implementing IS-IS for IPv6

In Cisco IOS Release 12.0(21)ST, Cisco IOS Release 12.0(22)S or later releases, and Cisco IOS Release 12.2(8)T or later releases, IS-IS support for IPv6 implements single-topology IPv6 IS-IS functionality based on IETF IS-IS WG *draft-ietf-isis-ipv6.txt*. A single shortest path first (SPF) per level is used to compute OSI, IPv4 (if configured) and IPv6 routes. The use of a single SPF means that both IPv4 IS-IS and IPv6 IS-IS routing protocols must share a common network topology. To use IS-IS for IPv4 and IPv6 routing, any interface configured for IPv4 IS-IS must also be configured for IPv6 IS-IS, and vice versa. All routers within an IS-IS area (Level 1 routing) or domain (Level 2 routing) must also support the same set of address families: IPv4 only, IPv6 only, or both IPv4 and IPv6.

Beginning with release Cisco IOS Release 12.2(15)T, Cisco IOS Release 12.2(18)S and Cisco IOS Release 12.3(1)M, IS-IS support for IPv6 is enhanced to also support multitopology IPv6 support as defined in IETF IS-IS WG *draft-ietf-isis-wg-multi-topology.txt*. Multitopology IPv6 IS-IS support uses multiple SPFs to compute routes and removes the restriction that all interfaces must support all configured address families and that all routers in an IS-IS area or domain must support the same set of address families.

The following IS-IS router configuration commands are specific to IPv4 and are not supported by, or have any effect on, IPv6 IS-IS:

`mpls`

`traffic-share`

If you are using IS-IS single-topology support for IPv6, IPv4, or both IPv6 and IPv4, you may configure both IPv6 and IPv4 on an IS-IS interface for Level 1, Level 2, or both Level 1 and Level 2. However, if both IPv6 and IPv4 are configured on the same interface, they must be running the same IS-IS level. That is, IPv4 cannot be configured to run on IS-IS Level 1 only on a specified ethernet interface while IPv6 is configured to run IS-IS Level 2 only on the same Ethernet interface.

5.1.3 Configuring Single-Topology IS-IS for IPv6

This task explains how to create an IPv6 IS-IS process and enable IPv6 IS-IS support on an interface.

Configuring IS-IS comprises two activities. The first activity, creates an IS-IS routing process using protocol-independent IS-IS commands. The second activity, configures the operation of the IS-IS protocol on an interface.

Command syntax for the global protocol-independent configuration:

To enable IS-IS for the specified IS-IS routing process, and enter router configuration mode use the following command:

```
Router(config)# router isis area-name
```

Next, configure an IS-IS network entity title (**net**) for the routing process:

```
Router(config-router)# net network-entity-title
```

The `network-entity-title` argument defines the area addresses for the IS-IS area and the system ID of the router.

After configuring an IS-IS routing process it may be enabled on appropriate interfaces by the following command:

```
Router(config-if)# ipv6 router isis area-name.
```

5.1.3.1 Examples

The following example shows the minimal configuration necessary to enable single topology IS-IS for IPv6 on a router. First the routing process is created, then this process is enabled on ethernet interface 0/0/1.

```
Router(config)# router isis area2
Router(config-router)# net 49.0001.0000.0000.000c.00
Router(config-router)# exit
Router(config)# interface Ethernet 0/0/1
Router(config-if)# ipv6 address 2001:0DB8::3/64
Router(config-if)# ipv6 router isis area2
Router(config-if)# exit
Router(config)#
```

5.1.4 Configuring Multitopology IS-IS for IPv6

This task explains how to configure multitopology IS-IS in IPv6. For these optional configuration steps it is assumed that IS-IS for IPv6 is already configured as described in the previous section. The commands described below may then be entered in the configuration context of the corresponding IS-IS routing process.

When multitopology IS-IS for IPv6 is configured, the **`transition`** keyword allows a user who is working with the single-topology SPF mode of IS-IS IPv6 to continue to work while upgrading to multitopology IS-IS. After every router is configured with the **`transition`** keyword, users can

remove the **transition** keyword on each router. When transition mode is not enabled, IPv6 connectivity between routers operating in single-topology mode and routers operating in multitopology mode is not possible.

You can continue to use the existing IPv6 topology while upgrading to multitopology IS-IS. The optional **isis ipv6 metric** command allows you to differentiate between link costs for IPv6 and IPv4 traffic when operating in multitopology mode.

Command syntax to enable multitopology IS-IS:

The following command configures a router running IS-IS to generate and accept only new-style TLVs:

```
Router(config-router)# metric-style wide [transition] \\
      [level-1 | level-2 | level-1-2]
```

Now, one enters IPv6 address family configuration mode.

```
Router(config-router)# address-family ipv6 [unicast]
```

The **unicast** keyword specifies the IPv6 unicast address family which is the default in this case anyway.

Within the IPv6 address family multi-topology IS-IS for IPv6 can be enabled with the following command:

```
Router(config-router-af)# multi-topology [transition]
```

The optional **transition** keyword allows an IS-IS IPv6 user to continue to use single-topology mode while upgrading to multitopology mode (s.a.).

5.1.4.1 Examples

In the following example a previously configured (single-topology) IS-IS session "area2" is switched to multi-topology IS-IS.

```
Router(config)# router isis area2
Router(config-router)# metric-style wide level-1
Router(config-router)# address-family ipv6
Router(config-router-af)# multi-topology
Router(config-router-af)# exit
Router(config-router)# exit
Router(config)#
```

5.1.5 Customizing IPv6 IS-IS

This task explains how to configure a new administrative distance for IPv6 IS-IS, configure the maximum number of equal-cost paths that IPv6 IS-IS will support, configure summary prefixes for IPv6 IS-IS, and configure an IS-IS instance to advertise the default IPv6 route (::/0). It also explains how to configure the hold-down period between partial route calculations (PRCs) and how often Cisco IOS software performs the SPF calculation when using multitopology IS-IS.

You can customize IS-IS multitopology for IPv6 for your network, but you likely will not need to do so. The defaults for this feature are set to meet the requirements of most customers and features. If

you change the defaults, refer to Cisco IPv4 configuration guide and IPv6 command reference to find the appropriate syntax.

Command syntax for customization of single-topology IS-IS for IPv6:

To inject the default route into an IS-IS routing domain the following command may be entered in the `address family ipv6` context of the corresponding IS-IS process:

```
Router(config-router-af)# default-information originate \\
      [route-map map-name]
```

The **`route-map`** keyword and `map-name` argument specify the conditions under which the IPv6 default route is advertised.

If the **`route-map`** keyword is omitted, then the IPv6 default route will be unconditionally advertised at Level 2.

The following command defines an administrative distance for IPv6 IS-IS routes in the IPv6 routing table. Like the previous command it is entered in the `address family ipv6` context.

```
Router(config-router-af)# distance value
```

The `value` argument is an integer from 10 to 254 (values 0 to 9 are reserved for internal use).

As with RIP (see section 4.1) and BGP (see section 7.1) one may define the maximum number of equal-cost routes that IPv6 IS-IS can support:

```
Router(config-router-af)# maximum-paths number-paths
```

Optionally one can allow a Level 1-2 router to summarize Level 1 prefixes at Level 2, instead of advertising the Level 1 prefixes directly when the router advertises the summary.

```
Router(config-router-af)# summary-prefix \\
      ipv6-prefix/prefix-length [level-1 | level-1-2 | level-2]
```

The `ipv6-prefix` argument in the **`summary-prefix`** command must be in the form documented in RFC 2373 where the address is specified in hexadecimal using 16-bit values between colons.

The argument is a decimal value that indicates how many high-order contiguous bits of the address comprise the prefix (the network portion of the address). A slash mark must precede the decimal value.

Last but not least, the hold-down period between PRCs and the interval between SPF calculations for multitopology IS-IS may also be manually set within the `address family ipv6` context:

```
Router(config-router-af)# prc-interval seconds \\
      [initial-wait] [secondary-wait]
Router(config-router-af)# spf-interval [level-1 | level-2] \\
      seconds [initial-wait] [secondary-wait]
```

The value of a multitopology IS-IS metric is not global but specific to an interface. Therefore it must be entered in that interface's configuration context:

```
Router(config-if) # isis ipv6 metric metric-value \\
      [level-1 | level-2 | level-1-2]
```

5.1.5.1 Examples

a) In the following example the IPv6 default route will be unconditionally advertised at Level 2:

```
Router(config)# router isis area2
Router(config-router)# address-family ipv6
Router(config-router-af)# default-information originate
```

b) This example shows the administrative distance of IPv6 IS-IS routes in this domain set to "90":

```
Router(config-router-af)# distance 90
```

c) In the following example all Level 1 routes belonging to the specified prefix are summarized at level 2:

```
Router(config-router-af)# summary-prefix 2001:0DB8::/24
```

d) This example shows the manual setting of IS-IS timers for IPv6:

```
Router(config-router-af)# prc-interval 20
Router(config-router-af)# spf-interval 30
```

e) For the last configuration example the routing context is exited and the configuration context for Ethernet interface 0/0/1 is entered to configure the metric for a multitopology IS-IS process on the routes from this interface:

```
Router(config-router-af)# exit
Router(config-router)# exit
Router(config)# interface Ethernet 0/0/1
Router(config-if)# isis ipv6 metric 20
```

5.1.6 Redistributing Routes

IPv6 Routes from another routing process/protocol can also be redistributed to an IPv6 IS-IS routing process. The corresponding **redistribute** command is entered in the `address family ipv6` context of the IS-IS routing process:

```
Router(config-router-af)# redistribute protocol [process-id] \\
      [level-1 | level-1-2 | level-2] [metric metric-value] \\
      [metric-type {internal | external}] [route-map map-name]
```

The `protocol` argument can be one of the following keywords: **bgp**, **connected**, **isis**, **rip** or **static**.

IPv6 routes learned on one IS-IS level may also be redistributed to another level of the same process. The corresponding **redistribute** command in this case includes the keyword **into**:

```
Router(config-router-af)# redistribute protocol \\
      [level-1 [into level-2] | level-1-2 | level-2 \\
      [into level-1]]
```

By default, routes learned by Level 1 instances are redistributed by Level 2 instance.

Note: The `protocol` argument must be **isis** in this configuration of the **redistribute** command.

5.1.6.1 Examples

a) In the following example `bgp` routes from `bgp` process `64500` are redistributed into the IS-IS routing domain with metric value:

```
Router(config-router-af)# redistribute bgp 64500 \\
      metric 100 route-map isismap
```

b) This example shows how IS-IS Level 1 routes are distributed in Level 2 domain:

```
Router(config-router-af)# redistribute isis \\
      level-1 into level-2
```

5.1.7 Disabling Consistency Checks

For single-topology IS-IS IPv6, routers must be configured to run the same set of address families. IS-IS performs consistency checks on hello packets and will reject hello packets that do not have the same set of configured address families. For example, a router running IS-IS for both IPv4 and IPv6 will not form an adjacency with a router running IS-IS for IPv4 or IPv6 only. In order to allow adjacency to be formed in mismatched address-families network, the **`adjacency-check`** command in IPv6 address family configuration mode must be disabled. This command is designed for use only in special situations.

This said, Cisco IOS software historically makes checks on hello packets to ensure that the IPv4 address is present and has a consistent subnet with the neighbor from which the hello packets are received. To disable this check, use the **`no adjacency-check`** command in the router configuration mode. However, if multitopology IS-IS is configured, this check is automatically suppressed, because multitopology IS-IS requires routers to form an adjacency regardless of whether or not all routers on a LAN support a common protocol.

Note: Disabling the **`adjacency-check`** command can adversely affect network configuration. Enter the **`no adjacency-check`** command only when you are running IPv4 IS-IS on all your routers and you want to add IPv6 IS-IS to your network but it's necessary to maintain all your adjacencies during the transition. When the IPv6 IS-IS configuration is complete, remove the **`no adjacency-check`** command from configuration.

5.1.8 Verifying IPv6 IS-IS Configuration, Operation and Debugging

After IS-IS configuration one might want to verify if everything works as it should. Cisco IOS offers a few commands that help accomplish this task:

The first (non IS-IS specific) command that can be used is:

```
Router# show ipv6 protocols [summary]
```

It displays the parameters and current state of the active IPv6 routing processes.

To view a list of all connected routers running IS-IS in all or specific areas use the following command:

```
Router# show isis [area-tag] [ipv6 | *] topology
```

The **show clns** command is used to display end system (ES), intermediate system (IS) and multitopology IS-IS (M-ISIS) neighbours.

```
Router# show clns [area-tag] neighbors \\
      [interface-type interface-number] [area] [detail]
```

The same command may also be used to display adjacency information for IS-IS neighbours:

```
Router# show clns area-name is-neighbors [type number] [detail]
```

The last command displays the IS-IS link-state database:

```
Router# show isis [area-tag] database [level-1] [level-2] \\
      [l1] [l2] [detail] [lspid]
```

You can use several system debugging commands to check your IS-IS for IPv6 implementation. If adjacencies are not coming up properly, use the **debug isis adj-packets** command.

```
Router# debug isis adj-packets
```

If you are using IS-IS multitopology for IPv6 and want to display statistical information about building routes between intermediate systems, use the **debug isis spf-statistics** command.

```
Router# debug isis spf-statistics
```

To display a log of significant events during an IS-IS SPF computation, use the **debug isis spf-events** command.

```
Router# debug isis spf-events
```

5.1.8.1 Examples

a) In the following example output information about the parameters and current state of that active IPv6 routing process is displayed using the `show ipv6 protocols` EXEC command:

```
Router# show ipv6 protocols
IPv6 Routing Protocol is "connected"
IPv6 Routing Protocol is "static"
IPv6 Routing Protocol is "isis"
Interfaces:
Ethernet0/0/3
Ethernet0/0/1
Serial1/0/1
Loopback1 (Passive)
Loopback2 (Passive)
Loopback3 (Passive)
Loopback4 (Passive)
Loopback5 (Passive)
Redistribution:
Redistributing protocol static at level 1
```

```
Address Summarization:
L2: 33::/16 advertised with metric 0
L2: 44::/16 advertised with metric 20
L2: 66::/16 advertised with metric 10
L2: 77::/16 advertised with metric 10
```

b) In the following example, output information about all connected routers running IS-IS in all areas is displayed using the **`show isis topology`** EXEC command:

```
Router# show isis topology
IS-IS paths to level-1 routers
System Id Metric Next-Hop Interface SNPA
0000.0000.000C
0000.0000.000D 20 0000.0000.00AA Se1/0/1 *HDLC*
0000.0000.000F 10 0000.0000.000F Et0/0/1 0050.e2e5.d01d
0000.0000.00AA 10 0000.0000.00AA Se1/0/1 *HDLC*
IS-IS paths to level-2 routers
System Id Metric Next-Hop Interface SNPA
0000.0000.000A 10 0000.0000.000A Et0/0/3 0010.f68d.f063
0000.0000.000B 20 0000.0000.000A Et0/0/3 0010.f68d.f063
0000.0000.000C --
0000.0000.000D 30 0000.0000.000A Et0/0/3 0010.f68d.f063
0000.0000.000E 30 0000.0000.000A Et0/0/3 0010.f68d.f063
```

c) In the following example, detailed output information that displays both end system (ES) and intermediate system (IS) neighbors is displayed using the **`show clns neighbors`** command with the **`detail`** keyword.

```
Router# show clns neighbors detail
System                                                  Id
Interface    SNPA            State  Holdtime  Type Protocol
0000.0000.0007     Et3/3        aa00.0400.6408  UP     26        L1
 IS-IS
Area Address(es): 20
IP Address(es): 172.16.0.42*
Uptime: 00:21:49
0000.0C00.0C35     Et3/2        0000.0c00.0c36  Up     91        L1
 IS-IS
Area Address(es): 20
IP Address(es): 192.168.0.42*
Uptime: 00:21:52
```

```
0800.2B16.24EA     Et3/3        aa00.0400.2d05  Up     27         L1
 M-ISIS
Area Address(es): 20
IP Address(es): 192.168.0.42*
IPv6 Address(es): FE80::2B0:8EFF:FE31:EC57
Uptime: 00:00:27
0800.2B14.060E     Et3/2        aa00.0400.9205  Up     8          L1
 IS-IS
Area Address(es): 20
IP Address(es): 192.168.0.30*
Uptime: 00:21:52
```

d) In the following example, output information to confirm that the local router has formed all the necessary IS-IS adjacencies with other IS-IS neighbors is displayed using the **`show clns is-neighbors`** EXEC command. To display the IPv6 link-local addresses of the neighbors, specify the **`detail`** keyword.

```
Router# show clns is-neighbors detail
System Id Interface State Type Priority Circuit Id Format
0000.0000.00AA Se1/0/1 Up L1 0 00 Phase V
Area Address(es): 49.0001
IPv6 Address(es): FE80::YYYY:D37C:C854:5
Uptime: 17:21:38
0000.0000.000F Et0/0/1 Up L1 64 0000.0000.000C.02 Phase V
Area Address(es): 49.0001
IPv6 Address(es): FE80::XXXX:E2FF:FEE5:D01D
Uptime: 17:21:41
0000.0000.000A Et0/0/3 Up L2 64 0000.0000.000C.01 Phase V
Area Address(es): 49.000b
IPv6 Address(es): FE80::ZZZZ:F6FF:FE8D:F063
Uptime: 17:22:06
```

e) In the following example, detailed output information about LSPs received from other routers and the IPv6 prefixes they are advertising is displayed using the **`show isis database`** EXEC command with the **`detail`** keyword specified:

```
Router# show isis database detail
IS-IS Level-1 Link State Database
LSPID LSP Seq Num LSP Checksum LSP Holdtime ATT/P/OL
0000.0C00.0C35.00-00 0x0000000C 0x5696 325 0/0/0
  Area Address: 47.0004.004D.0001
  Area Address: 39.0001
```

```
  Metric: 10 IS 0000.0C00.62E6.03
  Metric: 0 ES 0000.0C00.0C35
--More--
0000.0C00.40AF.00-00* 0x00000009 0x8452 608 1/0/0
  Area Address: 47.0004.004D.0001
  Topology: IPv4 (0x0) IPv6 (0x2)
  NLPID: 0xCC 0x8E
  IP Address: 172.16.21.49
  Metric: 10 IS 0800.2B16.24EA.01
  Metric: 10 IS 0000.0C00.62E6.03
  Metric: 0 ES 0000.0C00.40AF
  IPv6 Address: 2001:0DB8::/32
  Metric: 10   IPv6 (MT-IPv6) 2001:0DB8::/64
  Metric: 5    IS-Extended cisco.03
  Metric: 10   IS-Extended cisco1.03
  Metric: 10    IS (MT-IPv6) cisco.03
IS-IS Level-2 Link State Database:
LSPID LSP Seq Num LSP Checksum LSP Holdtime ATT/P/OL
0000.0000.000A.00-00 0x00000059 0x378A 949 0/0/0
Area Address: 49.000b
NLPID: 0x8E
IPv6 Address: 2001:0DB8:1:1:1:1:1:1
Metric: 10 IPv6 2001:2:YYYY::/64
Metric: 10 IPv6 3001:3:YYYY::/64
Metric: 10 IPv6 3001:2:YYYY::/64
Metric: 10 IS-Extended 0000.0000.000A.01
Metric: 10 IS-Extended 0000.0000.000B.00
Metric: 10 IS-Extended 0000.0000.000C.01
Metric: 0 IPv6 11:1:YYYY:1:1:1:1:1/128
Metric: 0 IPv6 11:2:YYYY:1:1:1:1:1/128
Metric: 0 IPv6 11:3:YYYY:1:1:1:1:1/128
Metric: 0 IPv6 11:4:YYYY:1:1:1:1:1/128
Metric: 0 IPv6 11:5:YYYY:1:1:1:1:1/128
0000.0000.000A.01-00 0x00000050 0xB0AF 491 0/0/0
Metric: 0 IS-Extended 0000.0000.000A.00
Metric: 0 IS-Extended 0000.0000.000B.00
```

5.2 Juniper JunOS

Since IS-IS for IPv6 is configured and treated in the same way as IPv4 this section only details the specific IPv6 commands and differences. For general IS-IS configuration guidelines please refer to Juniper's official JunOS documentation.

5.2.1 Configure IS-IS

To configure Intermediate System to Intermediate System (IS-IS), you include statements in the configuration:

```
protocols {
     isis  {
         disable ;
         ignore-attached-bit ;
         graceful-restart  {
            disable;
            helper-disable;
            restart-duration seconds ;
        }
         label-switched-path  name  level level  metric metric ;
         level  level-number  {
             authentication-key  key ;
             authentication-type  authentication ;
             external-preference  preference ;
            no-csnp-authentication;
            no-hello-authentication;
            no-psnp-authentication;
             preference  preference;
            prefix-export-limit number ;
             wide-metrics-only ;
        }
         lsp-lifetime  seconds ;
         no-authentication-check ;
        no-ipv4-routing;
        no-ipv6-routing;
         overload  <timeout seconds >;
         reference-bandwidth  reference-bandwidth;
         rib-group  group-name ;
         spf-delay  milliseconds ;
        topologies {
            ipv4-multicast;
```

```
        ipv6-unicast;
    }
     traffic-engineering  {
         disable ;
         shortcuts ;
    }
     traceoptions  {
        file name  <replace> <size size > <files  number > <no-stamp>
            <(world-readable | no-world-readable)>;
        flag flag  < flag-modifier > <disable>;
    }
     interface  interface-name  {
         disable ;
        bfd-liveness-detection {
            minimum-interval milliseconds ;
            minimum-receive-interval milliseconds ;
            minimum-transmit-interval milliseconds ;
            multiplier number ;
        }
        checksum;
         csnp-interval  ( seconds  | disable);
         lsp-interval  milliseconds ;
         mesh-group  ( value  | blocked);
        no-ipv4-multicast;
        no-ipv6-unicast;
         passive ;
        point-to-point;
         level  level-number  {
             disable ;
             hello-authentication-key  key ;
             hello-authentication-type  authentication ;
             hello-interval  seconds ;
             hold-time  seconds ;
            ipv4-multicast-metric number ;
            ipv6-unicast-metric number
             metric  metric ;
             passive ;
             priority  number ;
             te-metric  metric ;
```

```
            }
        }
    }
}
```

5.2.2 Disable IPv6 Routing

It is possible to disable Internet Protocol Version 6 (IPv6) routing for IS-IS. Disabling IPv6 routing results in the following:

- Router does not advertise the NLPID for IPv6 in JunOS software 0th LSP fragment.
- Router does not advertise any IPv6 prefixes in JunOS software LSPs.
- Router does not advertise the NLPID for IPv6 in JunOS software hello packets.
- Router does not advertise any IPv6 addresses in JunOS software hello packets.
- Router does not calculate any IPv6 routes.

To disable IPv6 routing on the router, include the **`no-ipv6-routing`** statement:

```
[edit protocols]
isis {
    no-ipv6-routing;
}
```

To re-enable IS-IS, remove the **`disable`** statement from the configuration:

```
[edit protocols]
user@host# delete isis no-ipv6-routing
```

5.2.3 Configure IS-IS IPv6 Unicast Topologies

You can configure IS-IS to calculate an alternate IPv6 unicast topology, in addition to the normal IPv4 unicast topology, and add the corresponding routes to `inet6.0`. The IS-IS interface metrics for the IPv4 topology can be configured independently of the IPv6 metrics. You can also selectively disable interfaces from participating in the IPv6 topology while continuing to participate in the IPv4 topology. This lets you exercise control over the paths that unicast data takes through a network.

To enable an alternate IPv6 unicast topology for IS-IS, include the **`ipv6-unicast`** statement:

```
[edit protocols]
isis {
    topologies {
        ipv6-unicast;
    }
}
```

To configure a metric for an alternate IPv6 unicast topology, include the **ipv6-unicast-metric** statement:

```
[edit protocols]
isis {
    interface interface-name {
        level level-number {
            ipv6-unicast-metric number;
        }
    }
}
```

To disable alternate IPv6 unicast topologies for IS-IS, include the **no-ipv6-unicast** statement:

```
[edit protocols]
isis {
    interface interface-name {
        no-ipv6-unicast;
    }
}
```

6 Implementing OSPF for IPv6

6.1 General Information about OSPFv3

OSPF is a routing protocol for IP. It is a link-state protocol, as opposed to a distance-vector protocol like RIP. Think of a link as being an interface on a networking device. A link-state protocol makes its routing decisions based on the states of the links that connect source and destination machines. The state of a link is a description of that interface and its relationship to its neighboring networking devices. The interface information includes the IPv6 prefix of the interface, the network mask, the type of network it is connected to, the routers connected to that network, and so on. This information is propagated in various type of link-state advertisements (LSAs).

A router's collection of LSA data is stored in a link-state database. The contents of the database, when subjected to the Dijkstra algorithm, result in the creation of the OSPF routing table. The difference between the database and the routing table is that the database contains a complete collection of raw data; the routing table contains a list of shortest paths to known destinations via specific router interface ports.

OSPF version 3, which is described in RFC 2740, supports IPv6.

Implementing OSPFv3 for IPv6 expands on OSPFv2 to provide support for IPv6 routing prefixes. This chapter describes the concepts and tasks you need to implement OSPFv3 for IPv6 on your network.

Note: OSPFv3 is a separate protocol from OSPFv2 and is therefore an IPv6-only routing protocol. If you are running a dual-stack environment with OSPF you need separate routing processes for IPv4 and IPv6 (OSPFv2 and OSPFv3 respectively).

6.1.1 LSA Types for IPv6

Due to the fact that with IPv6 it is possible to configure many different IP addresses on one interface LSA types for OSPv3 differ from those in OSPFv2 for IPv4.

The following list describes OSPFv3 LSA types, each of which has a different purpose:

- Router LSAs (Type 1)—Describes the link state and costs of a router's links to the area. These LSAs are flooded within an area only. The LSA indicates if the router is an Area Border Router (ABR) or Autonomous System Boundary Router (ASBR), and if it is one end of a virtual link. Type 1 LSAs are also used to advertise stub networks. In OSPF for IPv6, these LSAs have no address information and are network-protocol-independent. In OSPF for IPv6, router interface information may be spread across multiple router LSAs. Receivers must concatenate all router LSAs originated by a given router when running the SPF calculation.

- Network LSAs (Type 2)—Describes the link-state and cost information for all routers attached to the network. This LSA is an aggregation of all the link-state and cost information in the network. Only a designated router tracks this information and can generate a network LSA. In OSPF for IPv6, network LSAs have no address information and are network-protocol-independent.

- Interarea-prefix LSAs for ABRs (Type 3)—Advertises internal networks to routers in other areas (interarea routes). Type 3 LSAs may represent a single network or a set of networks summarized into one advertisement. Only ABRs generate summary LSAs. In OSPF for IPv6, addresses for these LSAs are expressed as *prefix, prefix length* instead of *address, mask*. The default route is expressed as a prefix with length 0.

- Interarea-router LSAs for ASBRs (Type 4)—Advertise the location of an ASBR. Routers that are trying to reach an external network use these advertisements to determine the best path to the next hop. ASBRs generate Type 4 LSAs.

- Autonomous system external LSAs (Type 5)—Redistributes routes from another AS, usually from a different routing protocol into OSPF. In OSPF for IPv6, addresses for these LSAs are expressed as *prefix, prefix length* instead of *address, mask*. The default route is expressed as a prefix with length 0.

- Link LSAs (Type 8)—Have local-link flooding scope and are never flooded beyond the link with which they are associated. Link LSAs provide the link-local address of the router to all other routers attached to the link, inform other routers attached to the link of a list of IPv6 prefixes to associate with the link, and allow the router to assert a collection of Options bits to associate with the network LSA that will be originated for the link.

- Intra-Area-Prefix LSAs (Type 9)—A router can originate multiple intra-area-prefix LSAs for each router or transit network, each with a unique link-state ID. The link-state ID for each intra-area-prefix LSA describes its association to either the router LSA or the network LSA and contains prefixes for stub and transit networks.

An address prefix occurs in almost all newly defined LSAs. The prefix is represented by three fields: PrefixLength, PrefixOptions, and Address Prefix. In OSPF for IPv6, addresses for these LSAs are expressed as *prefix, prefix length* instead of *address, mask*. The default route is expressed as a prefix with length 0. Type 3 and Type 9 LSAs carry all IPv6 prefix information that, in IPv4, is included in router LSAs and network LSAs. The Options field in certain LSAs (router LSAs, network LSAs, interarea-router LSAs, and link LSAs) has been expanded to 24 bits to provide support for OSPF in IPv6.

In OSPF for IPv6, the sole function of link-state ID in interarea-prefix LSAs, interarea-router LSAs, and autonomous-system external LSAs is to identify individual pieces of the link-state database. All addresses or router IDs that are expressed by the link-state ID in OSPF version 2 are carried in the body of the LSA in OSPF for IPv6.

The link-state ID in network LSAs and link LSAs is always the interface ID of the originating router on the link being described. For this reason, network LSAs and link LSAs are now the only LSAs whose size cannot be limited. A network LSA must list all routers connected to the link, and a link LSA must list all of the address prefixes of a router on the link.

6.1.2 NBMA in OSPF for IPv6

On NBMA networks, the designated router (DR) or backup DR (BDR) performs the LSA flooding. On point-to-point networks, flooding simply goes out an interface directly to a neighbor.

Routers that share a common segment (Layer 2 link between two interfaces) become neighbors on that segment. OSPF uses the Hello protocol, periodically sending hello packets out each interface.

Routers become neighbors when they see themselves listed in the neighbor's hello packet. After two routers become neighbors, they may proceed to exchange and synchronize their databases, which creates an adjacency. Not all neighboring routers have an adjacency.

On point-to-point and point-to-multipoint networks, the software floods routing updates to immediate neighbors. There is no DR or BDR; all routing information is flooded to each networking device.

On broadcast or NBMA segments only, OSPF minimizes the amount of information being exchanged on a segment by choosing one router to be a DR and one router to be a BDR. Thus, the routers on the segment have a central point of contact for information exchange. Instead of each router exchanging routing updates with every other router on the segment, each router exchanges information with the DR and BDR. The DR and BDR relay the information to the other routers.

The software looks at the priority of the routers on the segment to determine which routers will be DR and BDR. The router with the highest priority is the elected DR. If there is a tie, then the router with the higher router ID takes precedence. After the DR is elected, the BDR is elected the same way. A router with a router priority set to zero is ineligible to become the DR or BDR.

When using NBMA in OSPF for IPv6, you cannot automatically detect neighbors. On an NBMA interface, you must configure your neighbors manually using interface configuration mode.

6.2 Cisco IOS

6.2.1 Comparison of OSPF for IPv6 and OSPF Version 2

Much of the OSPF for IPv6 feature is the same as in OSPF version 2. OSPF version 3 for IPv6, which is described in RFC 2740, expands on OSPF version 2 to provide support for IPv6 routing prefixes and the larger size of IPv6 addresses.

In OSPF for IPv6, a routing process does not need to be explicitly created. Enabling OSPF for IPv6 on an interface will cause a routing process and its associated configuration, to be created.

Furthermore, in OSPFv3 on Cisco IOS, each interface must be enabled using commands in interface configuration mode. This feature is different from OSPF version 2, in which interfaces are indirectly enabled using the router configuration mode.

When using an nonbroadcast multiaccess (NBMA) interface in OSPF for IPv6, users must manually configure the router with the list of neighbors. Neighboring routers are identified by their router ID.

In IPv6, users can configure many address prefixes on an interface. In OSPF for IPv6, all address prefixes on an interface are included by default. Users cannot select some address prefixes to be imported into OSPF for IPv6; either all address prefixes on an interface are imported, or no address prefixes on an interface are imported.

Unlike OSPF version 2, multiple instances of OSPF for IPv6 can be run on a link.

6.2.2 Prerequisites

Before you enable OSPF for IPv6 on an interface, you should have an OSPF network strategy planned for your IPv6 network. For example, you must decide whether multiple areas are required.

On the routers on which you want to use OSPFv3 you need to have IPv6 unicast routing enabled and IPv6 address must be configured on the interfaces taking part in the routing process(es).

For the minimum required IOS version for features related to OSPFv3 routing please refer to:

http://www.cisco.com/univercd/cc/td/doc/product/software/ios123/123cgcr/ipv6_c/ftipv6s.htm#wp10 04964

6.2.3 Restrictions

Be careful when changing the defaults for commands used to enable OSPF for IPv6. Changing these defaults will affect your OSPF for IPv6 network, possibly adversely.

Authentication is not yet supported in all IOS releases. It is currently only available in 12.3(4)T. The feature will be more widely supported in OSPF for IPv6 in future releases.

6.2.4 Enabling OSPF for IPv6

To implement OSPFv3 on a Cisco IOS platform the protocol basically just needs to be enabled on an interface with an IPv6 address with the following command:

```
Router(config-if)# ipv6 ospf process-id \\
      area area-id [instance instance-id]
```

Now an OSPFv3 process with the specified id is created and the protocol is enabled. The next two sections on configuring NBMA interfaces and forcing SPF calculation are optional.

You can customize OSPF for IPv6 for your network, but you likely will not need to do so. The defaults for OSPF in IPv6 are set to meet the requirements of most customers and features. If you must change the defaults, refer to Cisco IPv4 configuration guide and Cisco IPv6 command reference to find the appropriate syntax. Be careful when changing the defaults.

6.2.4.1 Example

The following example configures a (new) OSPFv3 process with id 1 and area 0.

```
Router(config-if)# ipv6 ospf 1 area 0
```

6.2.5 Configuring NBMA Interfaces

You can customize OSPF for IPv6 in your network to use NBMA interfaces. Before you configure NBMA interfaces you must actually have your network configured as an NBMA network and identified each neighbor. You cannot automatically detect neighbours, but must manually configure your router to detect neighbours when using an NBMA interface. The commands to issue in the respective interface's configuration context are:

```
Router(config-if)# frame-relay map ipv6 ipv6-address dlci \\
      [broadcast] [cisco] [ietf] \\
      [payload-compression {packet-by-packet | \\
      frf9 stac [hardware-options] | data-stream stac \\
      [hardware-options]}]
```

The command defines the mapping between a destination IPv6 address and the data-link connection identifier (DLCI) used to connect to the destination address.

To configure a neighbour use the following command:

```
Router(config-if)# ipv6 ospf neighbor ipv6-address \\
[priority number] [poll-interval seconds] [cost number] \\
[database-filter all out]
```

6.2.5.1 Examples

a) In the following example, the NBMA link is frame relay. For other kinds of NBMA links, different mapping commands are used:

```
Router(config-if)#  frame-relay map ipv6 \\
     FE80::A8BB:CCFF:FE00:C01 120
```

b) The router on the frame relay link above is configured as an OSPFv3 neighbour:

```
Router(config-if) ipv6 ospf neighbor \\
      FE80::A8BB:CCFF:FE00:C01
```

6.2.6 Clearing the Databas and/or Forcing an SPF Calculation

Occasionally, when something was changed or for debugging purposes one might want to clear the present routing table and/or force the OSPFv3 process to build it a new. This is done using the following command, which is issued at the global configuration level:

```
Router # clear ipv6 ospf [process-id] {process | force-spf | \\
      redistribution | counters [neighbor [neighbor-interface]]}
```

The command clears the OSPF state based on the OSPF routing process ID.

When the **process** keyword is used with the **clear ipv6 ospf** command, the OSPF database is cleared and repopulated, and then the shortest path first (SPF) algorithm is performed. When the **force-spf** keyword is used with the **clear ipv6 ospf** command, the OSPF database is not cleared before the SPF algorithm is performed.

6.2.6.1 Example

The easiest way to do this is issuing the command without any more options:

```
Router# clear ipv6 ospf force-spf
```

6.2.7 Verifying OSPF for IPv6 Configuration and Operation

When debugging or verifying OSPFv3 configuration the **show ipv6 ospf** command comes in handy. It may be issued with the **interface** keyword, even adding a specific interface name for which OSPF-related information should be displayed. Or it may be used without an interface to display general OSPFv3 information:

```
Router# show ipv6 ospf [process-id] [area-id] \\
```

```
        interface [interface-type interface-number]
Router# show ipv6 ospf [process-id] [area-id]
```

6.2.7.1 Examples

a) This command displays OSPF-related inteface information:

```
Router# show ipv6 ospf interface
```

b) This command displays general information about OSPF routing processes:

```
Router# show ipv6 ospf
```

6.2.8 Load Balancing in OSPF for IPv6

When a router learns multiple routes to a specific network via multiple routing processes (or routing protocols), it installs the route with the lowest administrative distance in the routing table. Sometimes the router must select a route from among many learned from the same routing process with the same administrative distance. In this case, the router chooses the path with the lowest cost (or metric) to the destination. Each routing process calculates its cost differently and the costs may need to be manipulated in order to achieve load balancing.

OSPF performs load balancing automatically in the following way. If OSPF finds that it can reach a destination through more than one interface and each path has the same cost, it installs each path in the routing table. The only restriction on the number of paths to the same destination is controlled by the **`maximum-paths`** command. The default maximum paths is 16, and the range is from 1 to 64.

6.3 Juniper JunOS

Though OSPFv3 for IPv6 and OSPFv2 are separate and different protocols all of the configuration keywords available for both protocols are the same and have the same meaning. This section will therefore only focus on the differences in their configuration.

6.3.1 Configure OSPFv3

To configure OSPF Version 3, you include the following statements. Note that not all configuration statements available for IPv4 are present for IPv6 as well. Those that are however, are used in exactly the same way.

```
protocols {
    ospf3 {
        disable;
        export [ policy-names ];
        external-preference preference;
        graceful-restart {
            disable;
            helper-disable;
```

```
    notify-duration seconds;
    restart-duration seconds;
}
overload {
    <timeout seconds>;
}
preference preference;
prefix-export-limit;
reference-bandwidth reference-bandwidth;
rib-group group-name;
spf-delay;
traceoptions {
    file name <replace> <size size> <files number> <no-stamp>
        <(world-readable | no-world-readable)>;
    flag flag <flag-modifier> <disable>;
}
area area-id {
    area-range network/mask-length <restrict>;
    interface interface-name {
        disable;
        dead-interval seconds;
        hello-interval seconds;
        metric metric;
        passive;
        priority number;
        retransmit-interval seconds;
        transit-delay seconds;
    }
    nssa {
        area-range network/mask-length <restrict>;
        default-lsa {
            default-metric metric;
            metric-type type;
            type-7;
        }
        (no-summaries | summaries)
    }
    stub <default-metric metric> <summaries | no-summaries>;
    virtual-link neighbor-id router-id transit-area area-id {
```

```
                    disable;
                    dead-interval seconds;
                    hello-interval seconds;
                    retransmit-interval seconds;
                    transit-delay seconds;
                }
            }
        }
    }
```

6.3.1.1 Restrictions and Prerequisites

When you configure OSPFv3 on an interface, you must also include the **family inet6** statement at the [**edit interfaces** interface-name **unit** logical-unit-number] hierarchy level.

OSPFv3 does not support routing instances or authentication. It also only supports the no-forwarding and vrf routing instance types.

At least one router on each logical link must be eligible to be the designated router for OSPFv3.

For OSPFv3, one OSPF-specific interface must be created per interface name configured under OSPFv3. OSPFv3 does not allow interfaces to be configured by IP address.

Traffic engineering is not supported for OSPFv3.

6.3.2 Full Configuration Example

The following configuration was provided by FCCN, the portuguese NREN. In this case there is only one area, in which two of the router's interfaces are active.

```
ospfv3@junos# show policy-options:

policy-statement default6 {
    from {
        protocol [ static direct ];
        route-filter ::/0 exact;
    }
    then accept;
}

ospfv3@junos# show protocols:

ospf3 {
    export default6;
    area 0.0.0.0 {
        interface lo0.0;
        interface ge-0/0/0.5 {
            priority 80;
        }
        interface ge-0/2/0.2 {
            priority 80;
        }
    }
```

```
}


ospfv3@junos> show configuration interfaces ge-0/0/0 unit 5

vlan-id 5;
family inet {
    address 193.137.0.x/yy;
}
family iso {
    address 49.0001.0000.0000.0005.05;
}
family inet6 {
    address 2001:690:xxxx:1::1/64;
}
```

6.4 Quagga

OSPFv3 is a special daemon "`ospf6d`" within Quagga. To configure OSPFv3 via the command line one has to enter the OSPFv3 configuration context:

```
Router(config)# router ospf6 {}
```

The next step is to configure a **`router-id`**:

```
Router(config-rtr)# router-id <a.b.c.d> {}
```

After this step the interfaces, which are to participate in OSPFv6 routing need to be bound to the OSPFv3 routing process:

```
Router(config-rtr)# interface <ifname> area <area> {}
```

Zebra/Quagga will then start sending OSPF packets on this interface. `area` can be specified as 0.

6.4.1 Specific OSPFv3 Interface commands

Enter the configuration context for the interface you want to configure.

Command Syntax:

```
Router(config-if)#  ipv6 ospf6 cost <cost> {}
```

Sets interface's output cost (default value is 1).

Command Syntax:

```
Router(config-if)# ipv6 ospf6 hello-interval <hellointerval> {}
```

Sets interface's Hello Interval (default 40).

Command Syntax:

```
Router(config-if)# ipv6 ospf6 dead-interval <Ddeadinterval> {}
```

Sets interface's Router Dead Interval (default value is 40).

Command Syntax:

```
Router(config-if)# ipv6 ospf6 retransmit-interval <retransmitinterval> {}
```

Sets interface's Rxmt Interval (default value is 5).

Command Syntax:

```
Router(config-if)# ipv6 ospf6 priority <priority> {}
```

Sets interface's Router Priority (default value is 1).

Command Syntax:

```
Router(config-if)# ipv6 ospf6 transmit-delay <transmitdelay> {}
```

Sets interface's Inf-Trans-Delay (default value is 1).

6.4.2 Route Redistribution

OSPFv3 on the Quagga platform offers the possibility to redistribute static, connected or RIPng routes. The corresponding commands are entered in the **ospf6** area conext of the corresponding routing process.

```
Router(config-rtr)# redistribute static {}
Router(config-rtr)# redistribute connected {}
Router(config-rtr)# redistribute ripng {}
```

6.4.3 Displaying OSPFv3 Information

Several show commands can be used to display OSPFv3 routing information to verify correct configuration and router behaviour.

Command Syntax:

```
Router# show ipv6 ospf6 [<instance-id>] {}
```

`instance-id` is an optional OSPF instance ID. To see router ID and OSPF instance ID, simply type "**show ipv6 ospf6** ".

Command Syntax:

```
Router# show ipv6 ospf6 database {}
```

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

Command Syntax:

```
Router# show ipv6 ospf6 interface {}
```

Use this command to see OSPF interface configuration like for example costs.

Command Syntax:

```
Router# show ipv6 ospf6 neighbor {}
```

Shows the state and chosen (Backup) DR of a neighbor.

Command Syntax:

```
Router# show ipv6 ospf6 request-list <ipv6 address> {}
```

Shows the **`request-list`** of a neighbor.

Command Syntax:

```
Router# show ipv6 route ospf6 {}
```

This command shows the internal (OSPF) routing table.

7 Implementing Multiprotocol BGP for IPv6

This module describes how to configure Multiprotocol Border Gateway Protocol (BGP) for IPv6. BGP is an Exterior Gateway Protocol (EGP) used mainly to connect separate routing domains that contain independent routing policies (autonomous systems). Connecting to a service provider for access to the Internet is a common use for BGP. BGP can also be used within an autonomous system and this variation is referred to as internal BGP (iBGP). Multiprotocol BGP is an enhanced BGP that carries routing information for multiple network layer protocol address families, for example, IPv6 address family and for IP multicast routes. All BGP commands and routing policy capabilities can be used with multiprotocol BGP.

7.1 Cisco IOS

Multiprotocol BGP is the supported Exterior Gateway Protocol (EGP) for IPv6. Multiprotocol BGP extensions for IPv6 supports the same features and functionality as IPv4 BGP. IPv6 enhancements to multiprotocol BGP include support for an IPv6 address family and network layer reachability information (NLRI) and next hop (the next router in the path to the destination) attributes that use IPv6 addresses.

BGP uses a router ID to identify BGP-speaking peers. The BGP router ID is 32-bit value that is often represented by an IPv4 address. By default, the Cisco IOS software sets the router ID to the IPv4 address of a loopback interface on the router. If no loopback interface is configured on the router, then the software chooses the highest IPv4 address configured to a physical interface on the router to represent the BGP router ID. When configuring BGP on a router that is enabled only for IPv6 (the router does not have an IPv4 address), you must manually configure the BGP router ID for the router. The BGP router ID, which is represented as a 32-bit value using an IPv4 address syntax, must be unique to the BGP peers of the router.

7.1.1 Prerequisites

As with any IPv6 routing feature you must first enable **`ipv6 unicast-routing`** before configuring Multiprotocol BGP.

For the minimum required IOS version for multiprotocol BGP features please refer to:

http://www.cisco.com/univercd/cc/td/doc/product/software/ios123/123cgcr/ipv6_c/ftipv6s.htm#wp10 04964

7.1.2 Enabling and Implementing Multiprotocol BGP for IPv6

When configuring multiprotocol BGP extensions for IPv6, you must create the BGP routing process, configure peering relationships, and customize BGP for your particular network.

Note:The following sections describe the configuration tasks for creating an IPv6 multiprotocol BGP routing process and associating peers, peer groups, and networks to the routing process. The following sections do not provide in-depth information on customizing multiprotocol BGP because the protocol functions the same in IPv6 as it does in IPv4.

Command syntax:

```
Router(config)# router bgp autonomous-system-number
```

The command configures a BGP routing process, and enters router configuration mode for the specified routing process.

```
Router(config-router)# no bgp default ipv4-unicast
```

This command is used to disable the IPv4 unicast address family for the BGP routing process. Otherwise routing information for the IPv4 unicast address family is advertised by default.

As mentioned above usually an IPv4 address configured on one of the interfaces is used as BGP router id. You can optionally change/set this id with the following command:

```
Router(config-router)# bgp router-id ip-address
```

Note: Configuring a router id using the **bgp router-id** command resets all active BGP peering sessions.

As with IPv4 the next step is to configure BGP neighbors/peers. By default, neighbors that are defined using the **neighbor remote-as** command in router configuration mode exchange only IPv4 unicast address prefixes. To exchange other address prefix types, such as IPv6 prefixes, neighbors must also be activated using the **neighbor activate** command in address family configuration mode for the other prefix types, as shown for IPv6 prefixes.

The first step however is to add a neighbor in the context of previously created routing process:

```
Router(config-router)# neighbor ipv6-address remote-as \\
      autonomous-system-number
```

The specified `ipv6-address` may be a unicast IPv6 address of any form, but configuring IPv6 multiprotocol BGP between two IPv6 routers (peers) using link-local addresses requires that the interface for the neighbor be identified by using the **update-source** router configuration command, and that a route map be configured to set an IPv6 global next hop. The update-source is defined in the global BGP configuration context, after the neighbour was created:

```
Router(config-router)# neighbor ipv6-address update-source \\
      interface-type interface-number
```

If there are multiple connections to the neighbor and you do not specify the neighbor interface by using the `interface-type` and `interface-number` arguments in the **neighbor update-source** command, a TCP connection cannot be established with the neighbor using link-local addresses.

Now enter the IPv6 address family:

```
Router(config-router)# address-family ipv6 [unicast]
```

The **unicast** keyword specifies the IPv6 unicast address family. By default, the router is placed in configuration mode for the IPv6 unicast address family if the **unicast** keyword is not specified with the **address-family ipv6** command.

Within this context one can enable the neighbor to exchange prefixes for the IPv6 address family with the local router.

```
Router(config-router-af)# neighbor ipv6-address activate
```

By default, route maps that are applied in router configuration mode using the **neighbor route-map** command are applied to only IPv4 unicast address prefixes. Route maps for other address families must be applied in address family configuration mode using the **neighbor route-map** command, as shown for the IPv6 address family. The route maps are applied either as the inbound or outbound routing policy for neighbors under the specified address family. Configuring separate route maps

under each address family type simplifies managing complicated or different policies for each address family.

```
Router(config-router-af)# neighbor ipv6-address route-map \\
     map-name {in | out}
```

To configure a route map as it is needed when using a neighbour's link local address in the BGP configuration the special command **set ipv6 next-hop** is used in the route map configuration.

```
Router(config) # route-map map-name [permit | deny] \\
     [sequence-number]
Router(config-route-map)# match ipv6 address prefix-list \\
     prefix-list-name
Router(config-route-map)# set ipv6 next-hop ipv6-address \\
     [link-local-address]
```

The last command overrides the next hop advertised to the peer for IPv6 packets that pass a match clause of a route map for policy routing.

The `ipv6-address` argument specifies the IPv6 global address of the next hop. It doesn't need to be an adjacent router.

The `link-local-address` argument specifies the IPv6 link-local address of the next hop. It must be an adjacent router.

Note: The route map sets the IPv6 next-hop addresses (global and link-local) in BGP updates. If the route map is not configured, the next-hop address in the BGP updates defaults to the unspecified IPv6 address (::), which is rejected by the peer.

If you specify only the global IPv6 next-hop address (`ipv6-address` argument) with the **set ipv6 next-hop** command after specifying the neighbor interface (`interface-type` argument) with the **neighbor update-source**, the link-local address of the interface specified with the `interface-type` argument is included as the next-hop in the BGP updates. Therefore, only one route map that sets the global IPv6 next-hop address in BGP updates is required for multiple BGP peers that use link-local addresses.

As with IPv4 several BGP peers for which the same route maps and prefix lists apply can be configured as a peer group. This facilitates the configuration as well as optimizes the router's handling of the corresponding rules.

By default, peer groups that are defined in router configuration mode using the **neighbor peer-group** command exchange only IPv4 unicast address prefixes. To exchange other address prefix types, such as IPv6 prefixes, you must activate peer groups using the **neighbor activate** command in address family configuration mode for the other prefix types, as shown for IPv6 prefixes.

Members of a peer group automatically inherit the address prefix configuration of the peer group.

IPv4 active neighbors cannot exist in the same peer group as active IPv6 neighbors. Create separate peer groups for IPv4 peers and IPv6 peers.

Peer groups in general are defined in global BGP configuration mode. The following command creates a multiprocotol BGP peer group:

```
Router(config)# router bgp autonomous-system-number
Router(config-router)# neighbor peer-group-name peer-group
```

Activating this peer group for IPv6 neighbours and adding neighbours to it must be done in the IPv6 address family configuration context. One can first add neighbors to a peer group and then later activate all neighbours at once by activating the corresponding peer group.

```
Router(config-router)# address family ipv6 [unicast]
Router(config-router-af)# neighbor ipv6-address peer-group \\
      peer-group-name
Router(config-router-af)# neighbor {ip-address \\
      | peer-group-name | ipv6-address} activate
```

7.1.2.1 Examples

a) The following example enables IPv6 globally, configures a BGP process and establishes a BGP router ID. Also, the IPv6 multiprotocol BGP peer `2001:0DB8:0:CC00::` is configured and activated.

```
ipv6 unicast-routing
!
router bgp 65000
 no bgp default ipv4-unicast
 bgp router-id 192.168.99.70
 neighbor 2001:0DB8:0:CC00::1 remote-as 64600
address-family ipv6 unicast
 neighbor 2001:0DB8:0:CC00::1 activate
```

b) The following example configures the IPv6 multiprotocol BGP peer `FE80::XXXX:BFF:FE0E:A471` over Fast Ethernet interface 0 and sets the route map named `nh6` to include the IPv6 next-hop global address of Fast Ethernet interface 0 in BGP updates. The IPv6 next-hop link-local address can be set by the `nh6` route map (not shown in the following example) or from the interface specified by the **`neighbor update-source`** router configuration command (as shown in this example).

```
router bgp 65000
 neighbor FE80::XXXX:BFF:FE0E:A471 remote-as 64600
 neighbor FE80::XXXX:BFF:FE0E:A471 update-source fastethernet 0
address-family ipv6
 neighbor FE80::XXXX:BFF:FE0E:A471 activate
 neighbor FE80::XXXX:BFF:FE0E:A471 route-map nh6 out
route-map nh6 permit 10
 match ipv6 address prefix-list cisco
 set ipv6 next-hop 2001:5y6::1
ipv6 prefix-list cisco permit 2Fy2::/48 le 128
ipv6 prefix-list cisco deny ::/0
```

Note: If you specify only the global IPv6 next-hop address (`ipv6-address` argument) with the **`set ipv6 next-hop`** command after specifying the neighbor interface (`interface-type` argument) with the **`neighbor update-source`** command, the link-local address of the interface specified with the `interface-type` argument is included as the next hop in the BGP updates. Therefore, only one route map that sets the global IPv6 next-hop address in BGP updates is required for multiple BGP peers that use link-local addresses.

c) The following example configures the IPv6 multiprotocol BGP peer group named `group1`:

```
router bgp 65000
no bgp default ipv4-unicast
neighbor group1 peer-group
neighbor 2001:0DB8:0:CC00::1 remote-as 64600
address-family ipv6 unicast
 neighbor group1 activate
 neighbor 2001:0DB8:0:CC00::1 peer-group group1
```

d) The following example, configures the route map named `rtp` to permit IPv6 unicast routes from network `2001:0DB8::/24` if they match the prefix list named `cisco`:

```
router bgp 64900
no bgp default ipv4-unicast
neighbor 2001:0DB8:0:CC00::1 remote-as 64700
address-family ipv6 unicast
 neighbor 2001:0DB8:0:CC00::1 activate
 neighbor 2001:0DB8:0:CC00::1 route-map rtp in
ipv6 prefix-list cisco seq 10 permit 2001:0DB8::/24
route-map rtp permit 10
 match ipv6 address prefix-list cisco
```

7.1.3 Advertising Routes into IPv6 Multiprotocol BGP

This task explains how to advertise (inject) a prefix into IPv6 multiprotocol BGP.

By default, networks that are defined in router configuration mode using the **`network`** command are injected into the IPv4 unicast database. To inject a network into another database, such as the IPv6 BGP database, you must define the network using the **`network`** command in **`address family ipv6`** configuration.

```
Router(config-router-af)# network ipv6-address/prefix-length
```

The command advertises (injects) the specified prefix into the IPv6 BGP database. (The routes must first be found in the IPv6 unicast routing table.)

Routes are tagged from the specified prefix as "local origin."

7.1.3.1 Example

The following example injects the IPv6 network `2001:0DB8::/24` into the IPv6 unicast database of the local router. (BGP checks that a route for the network exists in the IPv6 unicast database of the local router before advertising the network.)

```
router bgp 65000
 no bgp default ipv4-unicast
address-family ipv6 unicast
network 2001:0DB8::/24
```

7.1.4 Redistributing Prefixes into IPv6 Multiprotocol BGP

This task explains how to redistribute (inject) prefixes from another routing protocol into IPv6 multiprotocol BGP.

To accomplish this task the **`redistribute`** command is used in the **`address-family ipv6`** configuration context of the corresponding BGP process.

```
Router(config-router-af)# redistribute protocol [process-id] \\
      [level-1 | level-1-2 | level-2] [metric metric-value] \\
      [metric-type {internal | external}] [route-map map-name]
```

The command specifies the routing protocol from which prefixes should be redistributed into IPv6 multiprotocol BGP.

The `protocol` argument can be one of the following keywords: **`bgp`**, **`connected`**, **`isis`**, **`rip`** or **`static`**.

Note: The **`connected`** keyword refers to routes that are established automatically by IPv6 having been enabled on an interface.

7.1.4.1 Example

The following example redistributes RIP routes into the IPv6 database of the local router:

```
router bgp 64900
no bgp default ipv4-unicast
address-family ipv6 unicast
 redistribute rip
```

7.1.5 Advertising IPv4 Routes Between IPv6 BGP Peers

This task explains how to advertise IPv4 routes between IPv6 peers. If an IPv6 network is connecting two separate IPv4 networks, it is possible to use IPv6 to advertise the IPv4 routes. Configure the peering using the IPv6 addresses within the IPv4 address family. Set the next hop with a static route or with an inbound route map because the advertised next hop will usually be unreachable. Advertising IPv6 routes between two IPv4 peers is also possible using the same model.

7.1.5.1 Examples

The following example advertises IPv4 routes between IPv6 peers when the IPv6 network is connecting two separate IPv4 networks. Peering is configured using IPv6 addresses in the IPv4 address family configuration mode. The inbound route map named `rmap` sets the next hop because the advertised next hop is likely to be unreachable.

```
router bgp 65000
!
neighbor 6peers peer-group
neighbor 2000:yyyy::2 remote-as 65002
address-family ipv4
neighbor 6peers activate
neighbor 6peers soft-reconfiguration inbound
neighbor 2000:yyyy::2 peer-group 6peers
neighbor 2000:yyyy::2 route-map rmap in
!
route-map rmap permit 10
set ip next-hop 10.21.8.10
```

7.1.6 Verifying IPv6 Multiprotocol BGP Configuration and Operation

Various show and debug commands may be used to verify IPv6 Multiprotocol BGP operation and configuration. The **show** command can be used as an unprivileged user.

Command syntax:

The following comannd displays entries in the IPv6 BGP routing table:

```
Router> show bgp ipv6 [ipv6-prefix/prefix-length] \\
      [longer-prefixes] [labels]
```

To view a summary of all BGP connections, use the following command:

```
Router> show bgp ipv6 summary
```

IPv6 BGP dampened routes may be displayed with this command:

```
Router> show bgp ipv6 dampened-paths
```

For the debug command one needs higher privilege levels.

```
Router# debug bgp ipv6 dampening [access-list-name] \\
      [prefix-list prefix-list-name]
```

The command displays debugging messages for IPv6 BGP dampening packets.

If no prefix list is specified, debugging messages for all IPv6 BGP dampening packets are displayed.

```
Router# debug bgp ipv6 updates [ipv6-address] \\
      [prefix-list prefix-list-name] [in | out]
```

The command can be used to display debugging messages for IPv6 BGP update packets.

If an `ipv6-address` argument is specified, debugging messages for IPv6 BGP updates to the specified neighbor are displayed.

Use the **`in`** keyword to display debugging messages for inbound updates only.

Use the **`out`** keyword to display debugging messages for outbound updates only.

7.1.6.1 Examples

a) In the following example, entries in the IPv6 BGP routing table are displayed using the **`show bgp ipv6`** user EXEC command:

```
Router> show bgp ipv6
BGP table version is 12612, local router ID is 192.168.99.70
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
*> 2001:0DB8:E:C::2 0 3748 4697 1752 i
*                   2001:0DB8:0:CC00::1
 0 1849 1273 1752 i
* 2001:618:3::/48 2001:0DB8:E:4::2 1 0 4554 1849 65002 i
*> 2001:0DB8:0:CC00::1
 0 1849 65002 i
*> 2001:620::/35 2001:0DB8:0:F004::1
  0 3320 1275 559 i
* 2001:0DB8:E:9::2 0 1251 1930 559 i
* 2001:0DB8::A  0 3462 10566 1930 559 i
* 2001:0DB8:20:1::11
  0 293 1275 559 i
* 2001:0DB8:E:4::2 1 0 4554 1849 1273 559 i
* 2001:0DB8:E:B::2 0 237 3748 1275 559 i
* 2001:0DB8:E:C::2 0 3748 1275 559 i
```

b) In the following example, the status of all IPv6 BGP connections is displayed using the **`show bgp ipv6 summary`** user EXEC command:

```
Router> show bgp ipv6 summary
BGP router identifier 192.168.7.225, local AS number 109
BGP table version is 21898, main routing table version 21898
175 network entries and 693 paths using 66927 bytes of memory
555 BGP path attribute entries using 29224 bytes of memory
539 BGP AS-PATH entries using 13620 bytes of memory
```

```
0 BGP route-map cache entries using 0 bytes of memory

0 BGP filter-list cache entries using 0 bytes of memory

Dampening enabled. 10 history paths, 31 dampened paths

BGP activity 2672/8496 prefixes, 21621/20928 paths, scan interval 15
secs

Neighbor V AS MsgRcvd MsgSent TblVer InQ OutQ Up/Down State/PfxRcd

2001:7yy:20:1::11     4     293   31525    14358 21898   0 0 14:29:34
    77

2001:Cyy:E:0:1::1     4     4768      0        0     0    0 0 never
 Active

FE80::2xx:4BFF:FE1A:E42

4 30 4442 4442 5 0 0 3d01h 3

FE80::2yy:6DFF:FE25:6000

4 20 4443 4444 5 0 0 3d01h 5
```

c) In the following example, IPv6 BGP dampened routes are displayed using the **`show bgp ipv6 dampened-paths`** user EXEC command:

```
Router> show bgp ipv6 dampened-paths

BGP table version is 12610, local router ID is 192.168.99.70

Status codes: s suppressed, d damped, h history, * valid, > best, i -
internal

Origin codes: i - IGP, e - EGP, ? - incomplete

Network From Reuse Path

*d 2001:0DB8::/24 2001:Cyy:E:B::2 00:00:10 237 2839 5609 i

*d 2001:2xx::/35 2001:Cyy:E:B::2 00:23:30 237 2839 5609 2713 i
```

d) In the following example, debugging messages for IPv6 BGP dampening packets are displayed using the **`debug bgp ipv6 dampening`** privileged EXEC command:

```
Router# debug bgp ipv6 dampening

00:13:28:BGP(1):charge penalty for 2000:y:0:1::/64 path 2 1 with
halflife-time 15 reuse/suppress 750/2000

00:13:28:BGP(1):flapped 1 times since 00:00:00. New penalty is 1000

00:13:28:BGP(1):charge penalty for 2000:y:0:1:1::/80 path 2 1 with
halflife-time 15 reuse/suppress 750/2000

00:13:28:BGP(1):flapped 1 times since 00:00:00. New penalty is 1000

00:13:28:BGP(1):charge penalty for 2000:y:0:5::/64 path 2 1 with
halflife-time 15 reuse/suppress 750/2000

00:13:28:BGP(1):flapped 1 times since 00:00:00. New penalty is 1000

00:16:03:BGP(1):charge penalty for 2000:y:0:1::/64 path 2 1 with
halflife-time 15 reuse/suppress 750/2000

00:16:03:BGP(1):flapped 2 times since 00:02:35. New penalty is 1892
```

```
00:18:28:BGP(1):suppress 2000:y:0:1:1::/80 path 2 1 for 00:27:30
(penalty 2671)

00:18:28:halflife-time 15, reuse/suppress 750/2000

00:18:28:BGP(1):suppress 2000:y:0:1::/64 path 2 1 for 00:27:20
(penalty 2664)

00:18:28:halflife-time 15, reuse/suppress 750/2000
```

e) In the following example, debugging messages for IPv6 BGP update packets are displayed using the **`debug bgp ipv6 updates`** privileged EXEC command:

```
Router# debug bgp ipv6 updates

14:04:17:BGP(1):2000:y:0:2::2 computing updates, afi 1, neighbor
version 0, table version 1, starting at ::

14:04:17:BGP(1):2000:y:0:2::2 update run completed, afi 1, ran for
0ms, neighbor version 0, start version 1, throttled to 1

14:04:19:BGP(1):sourced route for 2000:0:0:2::1/64 path #0 changed
(weight 32768)

14:04:19:BGP(1):2000:y:0:2::1/64 route sourced locally

14:04:19:BGP(1):2000:y:0:2:1::/80 route sourced locally

14:04:19:BGP(1):2000:y:0:3::2/64 route sourced locally

14:04:19:BGP(1):2000:y:0:4::2/64 route sourced locally

14:04:22:BGP(1):2000:y:0:2::2 computing updates, afi 1, neighbor
version 1, table version 6, starting at ::

14:04:22:BGP(1):2000:y:0:2::2 send UPDATE (format) 2000:0:0:2::1/64,
next 2000:0:0:2::1, metric 0, path

14:04:22:BGP(1):2000:y:0:2::2 send UPDATE (format) 2000:0:0:2:1::/80,
next 2000:0:0:2::1, metric 0, path

14:04:22:BGP(1):2000:y:0:2::2 send UPDATE (prepend, chgflags:0x208)
2000:0:0:3::2/64, next 2000:0:0:2::1, metric 0, path

14:04:22:BGP(1):2000:y:0:2::2 send UPDATE (prepend, chgflags:0x208)
2000:0:0:4::2/64, next 2000:0:0:2::1, metric 0, path
```

7.2 Juniper JunOS

To enable MBGP to carry network layer reachability information (NLRI) for IPv6 address family, include the family `inet6` statement:

```
family inet6 {
    (any | labeled-unicast | multicast | unicast) {
        prefix-limit {
            maximum number;
            teardown <percentage> <idle-timeout (forever | minutes)>;
        }
        rib-group group-name;
```

```
        }
    }
```

7.3 Quagga/Zebra

When adding IPv6 routing information exchange feature to BGP. There were some proposals. IETF IDR working group finally take a proposal called Multiprotocol Extension for BGP. The specification is described in RFC2283. The protocol does not define new protocols. It defines new attributes for existing BGP. When used to exchange IPv6 routing information it is called BGP-4+. When 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 does not have the feature to detect remote peer's capability whether it can handle other than IPv4 unicast routes. This is a big problem using Multiprotocol Extension for BGP to operational network. [RFC2842] proposing a feature called Capability Negotiation. `bgpd` use this Capability Negotiation to detect remote peer's capabilities. If the peer is only configured as IPv4 unicast neighbor, `bgpd` does not send these Capability Negotiation packets.

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 a remote peer please use `strict-capability-match` command.

Command Syntax:

```
Router# (no) neighbor <peer> strict-capability-match {}
```

This command strictly compares remote capabilities and local capabilities. If capabilities are different, an “Unsupported Capability error” is sent and the connection is finished.

You may want to disable sending capability negotiation `OPEN` message optional parameter to the peer when remote peer does not implement capability negotiation. Use `dont-capability-negotiate` command to disable this feature.

Command Syntax:

```
Router# (no) neighbor <peer> dont-capability-negotiate {}
```

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

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

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

Comannd Syntax:

```
Router# (no) neighbor <peer> override-capability {}
```

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

7.3.1 Sample BGP configuration

```
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
!
```

8 Implementing Security for IPv6

This chapter describes how to configure security features for IPv6. In the context of this cookbook security consists mainly on creating access control lists (ACLs) and configuring IPSec, but it also gives advice on how to securely operate on a router (i.e. by use of the secure shell remote login facility).

Note: There are two additional 6NET deliverables D3.5.1 [D3.5.1] and D3.5.1v2 [D3.5.1v2] ("Implementation of security plan version 1 & 2"), which may also serve as a source of information about topics not covered in this cookbook.

IPv6 ACL functionality is used for basic traffic filtering functions. Traffic filtering is based on source and destination addresses, inbound and outbound to a specific interface, and with an implicit deny statement at the end of each access list (functionality similar to standard ACLs in IPv4). In some implementations filtering traffic based on IPv6 option headers and optional, upper-layer protocol type information for finer granularity of control may also be available.

IPSec is a framework of open standards developed by the Internet Engineering Task Force (IETF) that provide security for transmission of sensitive information over unprotected networks such as the Internet. IPSec acts at the network layer, protecting and authenticating IP packets between participating IPSec devices (peers), such as Cisco routers. IPSec provides the following network security services. These services are optional. In general, local security policy will dictate the use of one or more of these services:

- Data confidentiality—The IPSec sender can encrypt packets before sending them across a network.
- Data integrity—The IPSec receiver can authenticate packets sent by the IPSec sender to ensure that the data has not been altered during transmission.
- Data origin authentication—The IPSec receiver can authenticate the source of the IPSec packets sent. This service is dependent upon the data integrity service.
- Antireplay—The IPSec receiver can detect and reject replayed packets.

Note: The term data authentication is generally used to mean data integrity and data origin authentication. Within this section it also includes antireplay services unless otherwise specified.

With IPSec, data can be sent across a public network without observation, modification or spoofing.

IPSec functionality is essentially identical in both IPv6 and IPv4; however, IPSec in IPv6 can be deployed from end-to-end—data may be encrypted along the entire path between a source node and destination node. (Typically, IPSec in IPv4 is deployed between border routers of separate networks.) In IPv6, IPSec is implemented using the authentication extension header and the ESP extension header. The authentication header provides integrity and authentication of the source. It also provides optional protection against replayed packets. The authentication header protects the integrity of most of the IP header fields and authenticates the source through a signature-based algorithm. The ESP header provides confidentiality, authentication of the source, connectionless integrity of the inner packet, antireplay and limited traffic flow confidentiality.

8.1 Router Login

One of the most important and basic security measures one should take is to secure access to all routing equipment. This is true for IPv6 as well as for IPv4. Secure Shell login is available today on most platforms, and often nothing has to be done to make this work also with IPv6 transport. In a few cases however, some special configuration or at least prerequisites are needed to enable IPv6 for SSH login.

8.1.1 Cisco IOS

This task describes how to enable SSH for use over an IPv6 transport. If you do not configure SSH parameters, then the default values will be used.

8.1.1.1 Prerequisites

Prior to configuring SSH over an IPv6 transport, ensure that the following conditions exist:

- An IPSec (Data Encryption Standard (DES) or 3DES) encryption software image from Cisco IOS Release 12.2(8)T or later releases or Cisco IOS Release 12.0(22)S or later releases is loaded on your router. IPv6 transport for the SSH server and SSH client requires an IPSec encryption software image.
- A host name and host domain are configured for the router. Refer to the "Mapping Host Names to IPv6 Addresses" section of the *Implementing Basic Connectivity for IPv6* module for information on assigning host names to IPv6 addresses and specifying default domain names that can be used by both IPv4 and IPv6.
- A Rivest, Shamir, and Adelman (RSA) key pair, which automatically enables SSH, is generated for the router. Refer to the "Configuring Certification Authority Interoperability" chapter of the Release 12.2 *Cisco IOS Security Configuration Guide* for information on generating an RSA key pair.

  Note: RSA is the public key cryptographic system developed by Ron Rivest, Adi Shamir, and Leonard Adelman. RSA keys come in pairs: one public key and one private key.
- A user authentication mechanism for local or remote access is configured on the router.

8.1.1.2 Restrictions

The basic restrictions for SSH over an IPv4 transport apply to SSH over an IPv6 transport. In addition to those restrictions, the use of locally stored user names and passwords is the only user authentication mechanism supported by SSH over an IPv6 transport; the TACACS+ and RADIUS user authentication mechanisms are not supported over an IPv6 transport.

Note: SSH clients can be authenticated by using TACACS+ or RADIUS over an IPv4 transport and then connecting to an SSH server over an IPv6 transport.

8.1.1.3 Configuration

The following command is used to configure **ssh** login:

```
Router(config)# ip ssh [timeout seconds] | \\
```

[**authentication-retries** *integer*]

The command configures SSH control variables on the router.

You can specify the timeout (in seconds) not to exceed 120 seconds. The default is 120. This setting applies to SSH negotiation phase. Once the EXEC session starts, the standard timeouts configured for the VTY apply.

By default, five VTY lines are defined (0-4); therefore, five terminal sessions are possible. After the SSH executes a shell, the VTY timeout starts. The VTY timeout defaults to 10 minutes.

You can also specify the number of authentication retries, not to exceed five authentication retries. The default is three.

8.1.1.4 Example

The example configures the **sshd** on the router with a timeout of 100 seconds and allows 2 retries for authentication:

```
Router(config)# ip ssh timeout 100 authentication-retries 2
```

8.2 IPv6 Access Control Lists

8.2.1 Cisco IOS

In Cisco IOS Release 12.0(23)S and 12.2(13)T or later releases, the standard IPv6 ACL functionality supports both traffic filtering based on source and destination addresses and filtering of traffic based on IPv6 option headers and optional, upper-layer protocol type information for finer granularity of control (functionality similar to extended ACLs in IPv4). IPv6 ACLs are defined using the **ipv6 access-list** command in global configuration mode and their permit and deny conditions are set using the **deny** and **permit** commands in IPv6 access list configuration mode. (Configuring the **ipv6 access-list** command places the router in IPv6 access list configuration mode, from which permit and deny conditions can be set for the defined IPv6 ACL.)

In Cisco IOS Release 12.0(23)S and 12.2(13)T or later releases, for backward compatibility, the **ipv6 access-list** command with the **deny** and **permit** keywords in global configuration mode is still supported. However, an IPv6 ACL defined with deny and permit conditions in global configuration mode is translated to IPv6 access list configuration mode.

IPv6 ACLs should be defined and then assigned to an interface. Otherwise, the interface operates with an empty access list until an access list is defined for the interface.

IPv6 ACLs are defined by a unique name (IPv6 does not support numbered ACLs). An IPv4 ACL and an IPv6 ACL cannot share the same name.

The IPv6 neighbor discovery process makes use of the IPv6 network layer service. By default, IPv6 ACLs implicitly allow IPv6 neighbor discovery packets to be sent and received on an interface. In IPv4, the Address Resolution Protocol (ARP), which is equivalent to the IPv6 neighbor discovery process, makes use of a separate data link layer protocol. Therefore, by default, IPv4 ACLs implicitly allow ARP packets to be sent and received on an interface.

In Cisco IOS Release 12.2(13)T and 12.0(23)S or later releases, every IPv6 ACL has implicit **permit icmp any any nd-na**, **permit icmp any any nd-ns**, and **deny ipv6 any any** statements as its last match conditions. (The former two match conditions allow for ICMPv6 neighbor discovery.) An

IPv6 ACL must contain at least one entry for the implicit **deny ipv6 any any** statement to take effect.

Time-based and reflexive ACLs are not supported for IPv4 or IPv6 on the Cisco 12000 series platform. The **reflect**, **timeout** and **time-range** keywords of the **permit** command in IPv6 may not be used on the Cisco 12000 series.

IPv6 access-lists are created in the global configuration context:

```
Router(config)# ipv6 access-list access-list-name
```

The command defines an IPv6 ACL and enters IPv6 access list configuration mode. The router prompt changes to `Router(config-ipv6-acl)#`.

The `access-list name` argument specifies the name of the IPv6 ACL. IPv6 ACL names cannot contain a space or quotation mark, or begin with a numeral.

Inside the configuration context for an IPv6 ACL a series of **permit** and **deny** statements may be submitted to specify **permit** and **deny** conditions:

```
Router(config-ipv6-acl)# permit {protocol} \\
      {source-ipv6-prefix/prefix-length | any | host \\
      source-ipv6-address} [operator [port-number]] \\
      {destination-ipv6-prefix/prefix-length | any \\
      | host destination-ipv6-address} [operator \\
      [port-number]] [dscp value] [flow-label value] \\
      [fragments] [log] [log-input] [reflect name \\
      [timeout value]] [routing] [time-range name] \\
      [sequence value]


Router(config-ipv6-acl)# deny {protocol} \\
      {source-ipv6-prefix/prefix-length | any | host \\
      source-ipv6-address} [operator [port-number]] \\
      {destination-ipv6-prefix/prefix-length | any \\
      | host destination-ipv6-address} [operator \\
      [port-number]] [dscp value] [flow-label value] \\
      [fragments] [log] [log-input] [routing] \\
      [time-range name] [undetermined transport] \\
      [sequence value]
```

The `protocol` argument specifies the name or number of an Internet protocol. It can be one of the keywords **ahp**, **esp**, **icmp**, **ipv6**, **pcp**, **sctp**, **tcp**, or **udp**, or an integer in the range from 0 to 255 representing an IPv6 protocol number.

The `source-ipv6-prefix/prefix-length` and `destination-ipv6-prefix/prefix-length` arguments specify the source and destination IPv6 network or class of networks about which to set permit conditions.

The **any** keyword is an abbreviation for the IPv6 prefix ::/0.

The **host** source-ipv6-address keyword and argument specify the source IPv6 host address about which to set permit conditions.

8.2.1.1 Controlling Access to a VTY

Filtering incoming and outgoing connections to and from the router based on an IPv6 ACL is performed using the **ipv6 access-class** command in line configuration mode. The **ipv6 access-class** command is similar to the **access-class** command except that the IPv6 ACLs are defined by a unique name, not a number. If the IPv6 ACL is applied to inbound traffic, the source address in the ACL is matched against the incoming connection source address and the destination address in the ACL is matched against the local router address on the interface. If the IPv6 ACL is applied to outbound traffic, the source address in the ACL is matched against the local router address on the interface and the destination address in the ACL is matched against the outgoing connection source address. We recommend that identical restrictions are set on all the virtual terminal lines because a user can attempt to connect to any of them.

IPv6 ACLs for this task are configured as described above in section 8.2.1.3. After having performed this task one enters the configuration context for the virtual terminal line to which access should be restricted:

```
Router(config)#  line [aux | console | tty | vty] \\
      line-number [ending-line-number]
```

The next step is to specify the ACL(s) which should be used to limit acces:

```
Router(config-line)# ipv6 access-class \\
      ipv6-access-list-name {in | out}
```

The ipv6-access-list name argument specifies the name of the IPv6 ACL. IPv6 ACL names cannot contain a space or quotation mark, or begin with a numeral.

8.2.1.2 Verifying IPv6 Security Configuration and Operation

This task explains how to display information to verify the configuration and operation of IPv6 security options. As usual this task involves special show and debug command and in this case occasionally also resetting the IPv6 access list match counters.

Command Syntax:

```
Router> show ipv6 access-list [access-list-name]
```

This **show** command displays the contents of all current IPv6 access lists.

Command Syntax:

```
Router> clear ipv6 access-list [access-list-name]
```

This command is used to reset the IPv6 access list match counters. If the access-list-name argument is specified, only the specified access list counters will be reset.

Command Syntax:

```
Router> debug ipv6 packet [access-list access-list-name] [detail]
```

This command displays debugging messages for IPv6 packets. If the **access-list** keyword and access-list-name argument is specified, only packets matching the access list permit entries are displayed.

8.2.1.3 Examples

a) The following example is from a router running Cisco IOS Release 12.2(13)T.

The example configures two IPv6 ACLs named OUTBOUND and INBOUND and applies both ACLs to outbound and inbound traffic on Ethernet interface 0. The first and second permit entries in the OUTBOUND list permit all TCP and User Datagram Protocol (UDP) packets from network 2001:yyyy:0300:0201::/64 to exit out of Ethernet interface 0. The entries also configure the temporary IPv6 reflexive ACL named REFLECTOUT to filter returning (incoming) TCP and UDP packets on Ethernet interface 0. The first deny entry in the OUTBOUND list keeps all packets from the network fec0:0:0:0201::/64 (packets that have the site-local prefix fec0:0:0:0201 as the first 64 bits of their source IPv6 address) from exiting out of Ethernet interface 0.

The evaluate command in the INBOUND list applies the temporary IPv6 reflexive ACL named REFLECTOUT to inbound TCP and UDP packets on Ethernet interface 0. When outgoing TCP or UDP packets are permitted on Ethernet interface 0 by the OUTBOUND list, the INBOUND list uses the REFLECTOUT list to match (evaluate) the returning (incoming) TCP and UDP packets.

```
ipv6 access-list OUTBOUND
 permit tcp 2001:yyyy:0300:0201::/64 any reflect REFLECTOUT
 permit udp 2001:yyyy:0300:0201::/64 any reflect REFLECTOUT
 deny fec0:0:0:0201::/64 any
ipv6 access-list INBOUND
 evaluate REFLECTOUT
interface ethernet 0
 ipv6 traffic-filter OUTBOUND out
 ipv6 traffic-filter INBOUND in
```

Note: Given that a permit any any statement is not included as the last entry in the OUTBOUND or INBOUND ACL, only TCP and UDP packets matching the configured permit entries in the ACL and ICMP packets matching the implicit permit conditions in the ACL are permitted out of and in to Ethernet interface 0 (the implicit deny all condition at the end of the ACL denies all other packet types on the interface).

b) The following example can be run on a router running Cisco IOS Release 12.2(13)T or 12.0(23)S. The example configures HTTP access to be restricted to certain hours during the day, and to log any activity outside of the permitted hours.

```
time-range lunchtime
periodic weekdays 12:00 to 13:00
ipv6 access-list OUTBOUND
permit tcp any any eq www time-range lunchtime
deny tcp any any eq www log-input
permit tcp 2000:1::/64 any
permit udp 2000:1::/64 any
```

c) In the following example, incoming connections to the virtual terminal lines 0 to 4 are filtered based on the IPv6 access list named `cisco`.

```
ipv6 access-list cisco
 permit ipv6 host 2000:0:0:4::2/128 any
!
line vty 0 4
 ipv6 access-class cisco in
```

d) In the following example, the **`show ipv6 access-list`** user EXEC command is used to verify that IPv6 ACLs are configured correctly:

```
Router> show ipv6 access-list
IPv6 access list inbound
permit tcp any any eq bgp reflect tcptraffic (8 matches) sequence 10
permit tcp any any eq telnet reflect tcptraffic (15 matches) sequence
20
permit udp any any reflect udptraffic sequence 30
IPv6 access list tcptraffic (reflexive) (per-user)
permit tcp host 2002:2222:1::1 eq bgp host 2002:2222:1::2 eq 11000
timeout 300 (time         left 243) sequence 1
permit tcp host 2002:2222:1::1 eq telnet host 2002:2222:1::2 eq 11001
timeout 300         (time left 296) sequence 2
IPv6 access list outbound
evaluate udptraffic
evaluate tcptraffic
```

e) In the following example, the **`show logging`** user EXEC command is used to display logging entries that match the first line (sequence 10) of the access list named `tin`:

```
Router> show logging
```

```
00:00:36:    %IPV6-6-ACCESSLOGP:    list    tin/10    permitted    tcp
2000:1::1(11001) (Ethernet0/0) -> 2000:1::2(179), 1 packet
```

f) In the following example, the **`show ipv6 access-list`** user EXEC command is used to display some match counters for the access list named `tin`. Privileged EXEC mode is entered using the **`enable`** command (not shown), and the **`clear ipv6 access-list`** EXEC command is issued to reset the match counters for the access list named `tin`. The **`show ipv6 access-list`** EXEC command is used again to show that the match counters have been reset.

```
Router> show ipv6 access-list tin
IPv6 access list tin
permit tcp any any log-input (6 matches) sequence 10
permit icmp any any echo-request log-input sequence 20
permit icmp any any echo-reply log-input sequence 30
Router# clear ipv6 access-list tin
Router# show ipv6 access-list tin
IPv6 access list tin
permit tcp any any log-input sequence 10
permit icmp any any echo-request log-input sequence 20
permit icmp any any echo-reply log-input sequence 30
```

8.3 Linux IP6tables

IP6tables is used to set up, maintain, and inspect the tables of IPv6 packet filter rules in the Linux kernel and is the IPv6 equivalent to IPtables. Several different tables may be defined. Each table contains a number of built-in chains and may also contain user-defined chains.

Each chain is a list of rules which can match a set of packets. Each rule specifies what to do with a packet that matches. This is called a `target', which may be a jump to a user-defined chain in the same table.

Currently standard IP6tables only allows packet filtering and packet mangling but connection tracking (stateful firwalling) are already being developed by USAGI and are available as patches since 2004. See section 8.3.3 for details.

This section covers IP6tables, it’s installation and basic/manual setup/configuration as well as firewall applications that build on IP6tables for Linux and embedded Linux systems.

8.3.1 IP6tables in General

General functionality for IP6tables is installed by compiling it into the Linux kernel or as a module. To make use of this one will also need the “ip6tables” script.

8.3.1.1 IP6tables in Linux 2.6

For Linux 2.6 kernels those features can be found in “Networking Options – Network Packet Filtering – IPv6: Netfilter Configuration” when configuring the kernel via Menuconfig or Xconfig. Alternatively one can just edit the configuration file “.config” in the kernel sources and set the following options to either “y” or “m” depending on whether a feature should be compiled into the kernel or as a module.

```
IP6_NF_IPTABLES =
```

General IP6tables support: This feature needs to be compiled whenever one wants to do any IPv6 packet filtering, masquerading or NATing. The following list contains suboptions to the IP6tables. It currently still expands/changes with practically every new version of the kernel.

```
CONFIG_IP6_NF_MATCH_LIMIT =
```

Limit match support: This feature allows to control the rate at which a rule can be matched. This is mainly useful in combination with the LOG target (“LOG target support” below) and to avoid some Denial of Service attacks.

```
CONFIG_IP6_NF_MATCH_RT =
```

Routing Header match support: Match packets based on the routing header.

```
CONFIG_IP6_NF_MATCH_MAC =
```

MAC address match support: Match packets based on the source ethernet address.

```
CONFIG_IP6_NF_MATCH_OPTS =
```

Hop-by-Hop and Destination-Options Header match support: Match packets based on the Hop-by-Hop or Destination-Options headers.

```
CONFIG_IP6_NF_MATCH_FRAG =
```

Fragmentation header match support: Match packets based on the fragmentation header.

```
CONFIG_IP6_NF_MATCH_HL =
```

Hop Limit match support: Math packets based on the hop limit.

```
CONFIG_IP6_NF_MATCH_MULTIPORT =
```

Multiple port match support: Multiport matching allows to match TCP or UDP packets based on a series of source or destination ports: normally a rule can only match a single range of ports.

```
CONFIG_IP6_NF_MATCH_OWNER =
```

Owner match support: Packet owner matching allows to match locally generated packets based on who created them: the user, group, process or session.

```
CONFIG_IP6_NF_MATCH_MARK =
```

Netfilter MARK match support: This feature allows to match packets based on the *nfmark* value in the packet. This can be set by the MARK target (see below).

```
CONFIG_IP6_NF_MATCH_IPV6HEADER =
```

IPv6 extension headers match: Match packets based upon IPv6 extension headers.

```
CONFIG_IP6_NF_MATCH_AHESP =
```

AH/ESP match support: Match AH and ESP packets.

```
CONFIG_IP6_NF_MATCH_LENGTH =
```

Packet Length match support: Match the Length of a packet against a specific value or range of values.

```
CONFIG_IP6_NF_MATCH_EUI64 =
```

EUI64 address check: This module performs checking on the IPv6 source address. It compares the last 64 bits with the EUI64 (delivered from MAC address) address.

```
CONFIG_IP6_NF_FILTER =
```

Packet filtering: Packet filtering defines a table *filter*, which has a series of rules for simple packet filtering at local input, forwarding and local output. It's the basic feature that gives the possibility to use `iptables(8)` (See below).

```
CONFIG_IP6_NF_TARGET_LOG =
```

LOG target support: This is a suboption to the packet filtering option. It adds a *LOG* target, which allows to create rules in any iptables table to record the packet header to syslog.

```
CONFIG_IP6_NF_MANGLE =
```

Packet mangling: This option adds a *mangle* table to iptables (see below for more information on iptables). This table is used for various packet alterations, which can effect how the packet is routed (including *NAT*).

```
CONFIG_IP6_NF_TARGET_MARK =
```

Raw table support (required for TRACE): This option adds a *raw* table to ip6tables. This table is the very first in the netfilter framework and hooks in at the PREROUTING and OUTPUT chains.

8.3.2 IP6wall

IP6wall is a bash script for ip6tables(8), it works fine with the 2.4.x series of the Linux kernel, but it should also work with linux 2.6.x. It was developed based on the source from the IPv4 iptables script gShield (http://muse.linuxmafia.org/gshield/). It was developed by Fabio Borraccetti (miasma@no-net.org) and is available under the General Public License (GPL), for example at:

http://www.no-net.org/ip6wall/download/ or http://freshmeat.net/projects/ip6wall/

8.3.2.1 Example configuration options

If one wants to log any firewall activities with the syslog service, set the following directive to "`yes`". The default is "`false`".

```
SYSLOG="false"
```

For the following directive to have any meaning one needs to have log rate limiting compiled into the kernel. The default is 20 per Minute ("`20/m`").

```
LTIME="20/m"
```

If reserved addresses (like multicast addresses) should not be logged, set following directive to "`NO`". The default is "`YES`".

```
LOG_RESERVED="YES"
```

The default is to have the default policy to log dropped packets. Otherwise, set `DEFAULT_LOGGING` to "`NO`".

```
DEFAULT_LOGGING=
```

You can set the log level at which ip6wall sends logging information to syslog. Default is "`7`" and should work fine.

```
LOG_LEVEL="7"
```

To which interfaces does outside traffic come from? (Incoming traffic)

```
EXTIF="sit0"
EXTIF2=
```

Which interface is local (connects to internal LAN/MAN)?

```
LOCALIF="eth0"
```

With the following directive you can specify the prefix your network is using and with it permit all traffic from and to destinations within this prefix. Leave this empty, if you don't want such a general rule.

```
LOCALNET="2002:638:500:200:0/64"
```

You can also add some external networks and addresses which you trust.

```
EXTERNALIP="2001:638:500:101::21/128"
EXTERNALIP2="2001:610:240:0:53::3/128"
```

The following directives tune the forwarding behaviour of the firewall. If `ALLOW_FORWARD` is set to "`YES`", traffic from inside the network (local) can be forwarded through the firewall. If `FORWARD_CLASS` is left empty, the firewall assumes that local traffic is all that originates within `LOCALNET`.

```
ALLOW_FOWARD="YES"
FORWARD_CLASS=""
```

With the DNS directive you can list your dns servers. If set to "`AUTO`", ip6wall will take the values from `/etc/resolv.conf`.

```
DNS="AUTO"
```

Set the default TCP and UDP response. It can be "`DROP`", "`REJECT`" or "`RESET`". The default is "`REJECT`" in both cases.

```
TCP_RESPONSE="REJECT"
UDP_RESPONSE="DROP"
```

Set the following directive to "`YES`", if you want to allow all ICMP traffic. The default is "`no`".

```
ICMP_ALLOW_ALL="NO"
```

Even if you allow all ICMP, ip6wall sets "sane" limits on how it goes about accepting ICMP. The default is 1 per second, but you can change that with the following variable.

```
ICMP_RATE="60/m"
```

If you want ICMP drops logged, set the following variable to "`YES`".

```
ICMP_LOG="YES"
```

If you want to allow traceroutes to the firewall box, set `TRACE_ALLOW_ALL` to "`YES`". It's very difficult to completely stop traceroutes. Ip6wall blocks the "standard" approach if set to "`NO`" below.

```
TRACE_ALLOW_ALL="NO"
```

Ip6wall allows a host to be listed as "completely" trusted with respect to access. However, it is restrictive – you need *both* the IP of this host and the MAC address of its ethernet card.

Set the following variable to "`YES`" if you need an `ADMIN` host.

```
ADMIN_HOST="NO"


ADMIN_HOST_IP="3ffe:aaaa:bbbb:cccc:dddd:eeee:ffff:0000"
```

```
ADMIN_HOST_MAC="aa:bb:cc:dd:ee:ff"
```

Note that this is inappropiate if the admin host is separated from the host machine by a router.

Having your machine ensure its time is set correctly is a good thing. Set the following variable to "YES", if you want to allow time syncs to go through the firewall.

```
ALLOW_TIME="YES"
```

List the servers with this directive.

```
TIMESERVERS="ntp.join.uni-muenster.de"
```

By default, ip6wall drops all high port connections via the default policy. To allow *individual* clients access to high ports, one should add these hosts to `conf/highport_access` (for example, specific IRC BOTNEts, etc).

If you want to disable this highport protection, and allow highports to be accessible to the world, set `ALLOW_ALL_HIGHPORT` below to "YES".

```
ALLOW_ALL_HIGHPORT="NO"
```

To configure port-forwarding, edit the file `conf/forwards` correspondingly and set the following variable to "YES".

```
FORWARDING="NO"
```

You can set which services are *open* and accessible to *every* host, as well as set connections to these services to be forwarded to a host behind the firewall.

Remember, you can *also* set specific services to be accessible *only* to specific clients (see the README for more details on this).

By default ip6wall *trusts* internal clients (as listed in NATS) - you can thus set a service to NO, and still have internal access to that service.

For each service, you have three possible options: `OPEN`, `FORWARD` or `NO`.

- `OPEN`: open to the world;
- `FORWARD`: open and forward connections to defined host;
- `NO`: do not open that service.

```
FTP_SERVICE="OPEN"
FTP_HOST="IP_TO_FORWARD"

HTTP_SERVICE="OPEN"
HTTP_HOST="IP_TO_FORWARD"

HTTPS_SERVICE="NO"
HTTPS_HOST="IP_TO_FORWARD"

SSHD_SERVICE="OPEN"
SSHD_HOST="IP_TO_FORWARD"
```

```
SMTP_SERVICE="OPEN"
SMPT_HOST="IP_TO_FORWARD"

POP_SERVICE="NO"
POP_HOST="IP_TO_FORWARD"

IMAPD_SERVICE="NO"
IMAPD_HOST="IP_TO_FORWARD"

IMAPDSSL_SERVICE="NO"
IMAPDSSL_HOST="IP_TO_FORWARD"

TELNET_SERVICE="NO"
TELNET_HOST="IP_TO_FORWARD"

NFS_SERVICE="NO"
NFS_HOST="IP_TO_FORWARD"

FINGER_SERVICE="NO"
FINGER_HOST="IP_TO_FORWARD"

TIME_SERVICE="NO"
TIME_SERVICE_HOST="IP_TO_FORWARD"
```

Auth has the additional options of sending a tcp-reset back (`RETURN`) or a standard `REJECT` regardless of the default ip6wall response setting.

```
IDENTD_SERVICE="OPEN"
```

BIND options are somewhat different from the rest.

```
BIND_SERVICE="OPEN"
```

If BIND is running locally on the firewall, set the following variable to "`YES`".

```
LOCAL_BIND="YES"
```

If ip6wall should specifically forward bind connections set the following variable to "`YES`" and specify a destination host.

```
FORWARD_BIND="NO"
DNS_HOST="IP_TO_FORWARD"
```

If you want to allow zone transfers set the following directive to "`YES`" and make sure you are using proper ACLs for BIND.

```
ALLOW_ZONE_TRANSFERS="YES"
```

8.3.3 IPv6 connection tracking

For a few years already USAGI has been working on IPv6 connection tracking. In analogy to the IPv4 feature `IP_CONNTRACK` the module is called `IP6_CONNTRACK` and has been available for 2.5/6 kernels since 2001 with increasing amount of stability.

The corresponding option in the kernel configuration is:

```
CONFIG_IP6_NF_CONNTRACK =
```

Connection tracking keeps a record of what packets have passed through the machine, in order to figure out how they are related to connections. It can also be used to enhance packet filtering when used together with the following option:

```
CONFIG_IP6_NF_MATCH_STATE =
```

Connection state match support: Connection state matching allows to match packets based on their relationship to a tracked connection (ie. Previous packets). This is a powerful tool for packet classification and often used on, for example FTP servers.

`IP6_CONNTRACK` never made it into the official netfilter module because of heavy code replication with regards to the `IP_CONNTRACK` module. It was felt that this was bad design and a common generalized module should instead be developed that embraces both IPv4 and IPv6 and makes use of the similarities in both protocols. This goal has recently (October 2004) been reached and a new module `NF_CONNTRACK` has been designed to cover both IPv6 and IPv4. Nearly all functionality previously available in the two separate modules with a few restrictions yet (i.e. no IPv4 NAT).

In all the IPv6 connection tracking functionality is similar to the corresponding feature for IPv4. If it is installed you can add rules of the following form:

```
# ip6tables –A INPUT –j ACCEPT –m state –state ESTABLISHED,RELATED
```

8.4 PF Packet Filter Firewall (BSD)

The PF packet filter firewall originally developed for OpenBSD operating system to replace the ipfilter packet filtering firewall. The development of PF closely tied to OpenBSD, but it has been ported to FreeBSD and NetBSD. It's available for OpenBSD since version 3.0, for FreeBSD since version 5.3 and for NetBSD since version 2.0.

PF is a stateful packet filter, which means it can track the state of a connection. Instead of passing all traffic to a particular port, it is possible to pass only the initial packet, and then begin to keep state. Subsequent traffic will flow because the filter is aware of the connection or related packets (even for UDP and ICMP).

8.4.1 IPv6 support

The PF firewall system supports basic IPv6 filtering since it incarnation. It is very easy to setup IPv6 PF firewall rules next to the IPv4 rules, since only the address family '`inet6`' should be added next to address family '`inet`', taking into consideration the Neighbor Discovery setup as discussed in D3.5.1 [D3.5.1] and in [TNC2004_IPv6_FW].

The current version has several good IPv6 filtering capability for IPv6:

- ICMPv6 filtering capability with support of filtering based on ICMPv6 message type and message code.
- Support for ICMPv6 message pairing or stateful filtering (or connection tracking), that allows easier setup of IPv6 filtering rules based on an initial packet and implicitly permitting

answers to this ICMPv6 packet. This allows easier setup of Neighbor Solicitation/Neighbor Advertisement and Router Solicitation/Router Advertisement pairs.

- Path MTU discovery passing automatically supported if keep-state is used. The same applies for Destination Unreachable, Time Exceeded and IPv6 Parameter Problem ICMPv6 messages.
- Support of Checks for existence of particular IPv6 extension headers.
- IPv6 QoS possible with ALTQ framework

Some IPv6 problems of the current PF implementations:

- TCP fragments are not handled properly. They are currently unconditionally blocked, which is less than optimal.
- TCP fragments normalisaton is not possible - not possible since fragmentation and reassembly is allowed only at end-hosts.
- No IPv6 support in ftp-proxy.
- No support to filter inside tunnels, except if a tunnel terminated at the firewall.

Table 1 Supported ICMPv6 types

Symbolical Name	Code	Description
`unreach`	1	destination unreachable
`toobig`	2	packet too big
`timex`	3	time exceeded
`paramprob`	4	parameter problem
`echoreq`	128	echo request
`echorep`	129	echo reply
`listqry`	130	MLD listener query
`listenrep`	131	MLD listener report
`listendone`	132	MLD listener done
`routersol`	133	ND router solicitation
`routeradv`	134	ND router advertisement
`neighbrsol`	135	ND neighbor solicitation
`neighbradv`	136	ND neighbor advertisement
`redir`	137	ND redirection
`routerrenum`	138	ICMPv6 router renumbering
`nireq`	139	neighbor information query
`nirep`	140	neighbor information reply

mtraceresp	200	MLD multicast trace response
mtraceresp	201	MLD multicast trace

8.4.2 Example Setup 1

The following example shows, how you can setup IPv6 only rules. It denies any incoming traffic, and allows all outgoing DNS requests, TCP connections and ping request. The **EXT** represents the external interface of firewall, while **LAN** denotes the protected LAN interface of the firewall.

```
EXT = bge0"
LAN = bge1"
LANip6 = 2001:db8:1:1::1 # IPv6 address of LAN interface
EXTip6 = 2001:db8:1:2::1 # IPv6 address of external network interface
LANnet6 = 2001:db8:1:1::1/64 # IPv6 network address of the LAN
Lo6 = ::1 #IPv6 lookback interface
set optimization aggressive # expire state connections early
antispoof for lo0 inet6 # antispoof setup (BCP 0038)
antispoof for $LAN inet6 # antispoof setup (BCP 0038)
block in log all # block all incoming packets on both interface
# allow DNS requests to go out:
pass out on $EXT inet6 proto udp from {$EXTip6, $Lo6, $LANnet6} to any
port=domain keep state
# all TCP request allowed out:
pass out on $EXT inet6 proto tcp from {EXTip6, $Lo6, $LANnet6} to any
keep state
# all ping request allowed out:
pass out on $EXT inet6 proto icmp6 all icmp6-type echoreq keep state
# ND solicitation out:
pass out on $EXT inet6 proto icmp6 all icmp6-type {neighbradv, neighbrsol}
# ND advertisement in:
pass in on $EXT inet6 proto icmp6 all icmp6-type {neighbradv, neighbrsol}
# router advertisement out:
pass out on $LAN inet6 proto icmp6 all icmp6-type routersadv
# router solicitation in:
pass in on $LAN inet6 proto icmp6 all icmp6-type routersol
# DNS request inside:
pass in on $LAN inet6 proto from $LANnet6 to any port domain
# TCP request inside:
pass in on $LAN inet6 proto tcp from $LANnet6 to any
```

```
# ICMP request inside:
pass in on $LAN inet6 proto icmp6 all icmp6-type echoreq
```

8.4.3 Example Setup 2

The following example shows, how you can setup IPv6 only rules, deny any incoming traffic except to mail and WWW server, and allows all outgoing DNS request, TCP connections and ping request. The **EXT** represents the external interface of firewall, while **LAN** denotes the protected LAN interface of the firewall.

```
EXT = bge0"
LAN = bge1"
LANip6 = 2001:db8:1:1::1 # IPv6 address of LAN interface
EXTip6 = 2001:db8:1:2::1 # IPv6 address of external network interface
LANnet6 = 2001:db8:1:1::1/64 # IPv6 network address of the LAN
Lo6 = ::1 #I Pv6 lookback interface
LANSRV6=2001:db8:1:2::2 # internal server address
LANSRV4=192.168.1.2 # internal server address
set optimization aggressive # expire state connections early
antispoof for lo0 inet6 # antispoof setup (BCP 0038)
antispoof for $LAN inet6 # antispoof setup (BCP 0038)
block in log all # block all incoming packets on both interface
# allow DNS reques:
pass out on $EXT inet6 proto udp from {$EXTip6, $Lo6, $LANnet6} to any
port=domain keep state
# all TCP request allowed out:
pass out on $EXT inet6 proto tcp from {EXTip6, $Lo6, $LANnet6} to any keep
state
# all ping request allowed out:
pass out on $EXT inet6 proto icmp6 all icmp6-type echoreq keep state
# ND solicitation out:
pass out on $EXT inet6 proto icmp6 all icmp6-type {neighbradv, neighbrsol}
# ND advertisement in:
pass in on $EXT inet6 proto icmp6 all icmp6-type {neighbradv, neighbrsol}
# router advertisement out:
pass out on $LAN inet6 proto icmp6 all icmp6-type routeradv
# router solicitation in:
pass in on $LAN inet6 proto icmp6 all icmp6-type routersol
# DNS request inside:
pass in on $LAN inet6 proto from $LANnet6 to any port domain
```

```
# TCP request inside:
pass in on $LAN inet6 proto tcp from $LANnet6 to any
# ICMP request inside:
pass in on $LAN inet6 proto icmp6 all icmp6-type echoreq
# allow incoming connection to SMTP server:
pass in on $EXT inet6 proto tcp from any to $LANSRV6 port=25 keep-state
pass in on $EXT inet proto tcp from any to $LANSRV4 port=25 keep-state
# all reply from SMTP server (does not really necessary):
pass in on $LAN inet6 proto tcp from $LANSRV6 port=25 to any keep-state
pass in on $LAN inet proto tcp from $LANSRV4 port=25 to any keep-state
# allow incoming connection to WWW server:
pass in on $EXT inet6 proto tcp from any to $LANSRV6 port=www keep-state
pass in on $EXT inet proto tcp from any to $LANSRV4 port=www keep-state
# all reply from SMTP server (does not really necessary):
pass in on $LAN inet6 proto tcp from $LANSRV6 port=www to any keep-state
pass in on $LAN inet proto tcp from $LANSRV4 port=www to any keep-state
```

8.5 IPv6FW Packet Filter Firewall

The ip6fw is native IPv6 filtering interface since FreeBSD without stateful inspection. It requires `IPV6FIREWALL` and `IPV6FIREWALL_VERBOSE` enabled in the FreeBSD 4 based kernel for filtering and for logging respectively. It requires `PFIL-HOOKS` to be compiled in the kernel in case of FreeBSD 5. In case of FreeBSD 5.3 and later it does not require anything since `PFIL_HOOKS` is included by default.

There several limitations of ip6fw:

- Does not support header chaining.
- Does not have stateful inspection.
- Does not support ICMPv6 error codes.
- Does not support TCP flags on IPv6 packets.

However it has some advantages:

- It supports all necessary ICMPv6 packet types.
- It supports checking of IPv6 extension headers.
- It is supported in FreeBSD 4.x and FreeBSD 5.x.

8.5.1 Example Snippets For IPv6FW

```
# Allow DAD
ip6fw add pass ipv6-icmp from :: to ff02::/16
```

```
# Allow RA,RS, NS, NA and redirect
ip6fw add pass ipv6-icmp from fe80::/10 to fe80::/10
ip6fw add pass ipv6-icmp from fe80::/10 to ff02::/16

# Allow link-local multicast traffic
ip6fw add pass all from fe80::/10 to ff02::/16
ip6fw add pass all from ${net}/${prefixlen} to ff02::/16

# Allow ICMPv6 destination unreachable
ip6fw add pass ipv6-icmp from any to any icmptype 1

# Allow PATH-MTU  - do not filter!
ip6fw add pass ipv6-icmp from any to any icmptype 2

# Allow NS/NA - do not filter!
ip6fw add pass ipv6-icmp from any to any icmptype 135,136
```

9 Implementing IPv6 Multicast

Traditional IP and IPv6 communication allows a host to send packets to a single host (unicast transmission) or to all hosts (broadcast transmission). Multicast provides a third scheme, allowing a host to send a single data stream to a subset of hosts (group transmission) simultaneously.

This module describes the concepts and tasks needed to implement IPv6 multicast on the network.

9.1 IPv6 Multicast Concepts

The new PIM-SM IETF draft [PIM-SM] introduces apart of numerous protocol modifications clearer conceptual separation of all the functions necessary for the network to deliver multicast packets. Considering that IPv6 multicast is always very recent implementation, most of the vendors implement the new draft.

1) Multicast Routing Information Base (MRIB) - necessary to determine where are the sources of the multicast data and perform the Reverse Path Forwarding (RPF) check. The content of it are unicast routing tables produced by different routing protocols.

2) Tree Information Base (TIB) - state information created by PIM and MLD (Multicast Listener Discovery) procedures and kept by the router. This information is used to populate Multicast Forwarding Information Base (MFIB) to actually forward the data. The relationship between TIB and MFIB can be compared to the Routing Table and CEF (Cisco Express Forwarding) table in the case of ordinary unicast forwarding.

3) Multicast Forwarding Information Base (MFIB) - which takes purely care of data forwarding. This table is built using the MRIB and TIB information.

9.1.1 IPv6 Multicast Overview

An IPv6 multicast group is an arbitrary group of receivers that want to receive a particular data stream. This group has no physical or geographical boundaries - receivers can be located anywhere on the Internet or in any private network. Receivers that are interested in receiving data flowing to a particular group must join the group by signalling their local router. This signalling is achieved with the MLD protocol.

Routers use the MLD protocol to learn whether members of a group are present on their directly attached subnets. Hosts join multicast groups by sending MLD report messages. The network then delivers data to a potentially unlimited number of receivers, using only one copy of the multicast data on each subnet. IPv6 hosts are known as group members.

Packets delivered to group members are identified by a single multicast group address. Multicast packets are delivered to a group using best-effort reliability, just like IPv6 unicast packets.

The multicast environment consists of senders and receivers. Any host, regardless of whether it is a member of a group, can send to a group. However, only the members of a group receive the message.

A multicast address is chosen for the receivers in a multicast group. Senders use a multicast address as the destination address of a datagram to reach all members of the group.

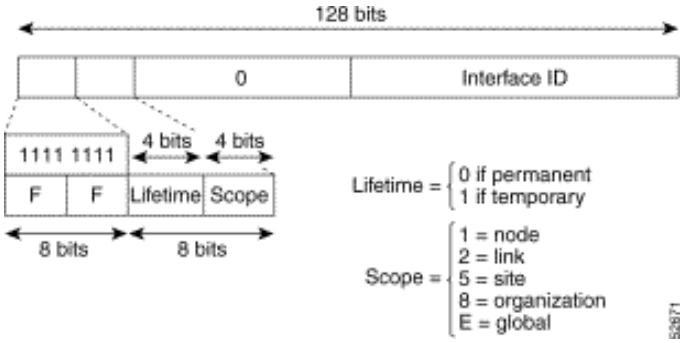
Membership in a multicast group is dynamic; hosts can join and leave at any time. There is no restriction on the location or number of members in a multicast group. A host can be a member of more than one multicast group at the same time.

How active a multicast group is, its duration, and its membership can vary from group to group and from time to time. A group that has members may have no activity.

9.1.2 IPv6 Multicast Addressing

An IPv6 multicast address is an IPv6 address that has a prefix of FF00::/8 (1111 1111). An IPv6 multicast address is an identifier for a set of interfaces that typically belong to different nodes. A packet sent to a multicast address is delivered to all interfaces identified by the multicast address. The second octet following the prefix defines the lifetime and scope of the multicast address. A permanent multicast address has a lifetime parameter equal to 0; a temporary multicast address has a lifetime parameter equal to 1. A multicast address that has the scope of a node, link, site, or organization, or a global scope has a scope parameter of 1, 2, 5, 8, or E, respectively. For example, a multicast address with the prefix FF02::/16 is a permanent multicast address with a link scope. Figure 1 shows the format of the IPv6 multicast address.

Figure 1: IPv6 Multicast Address Format



IPv6 nodes (hosts and routers) are required to join (receive packets destined for) the following multicast groups:

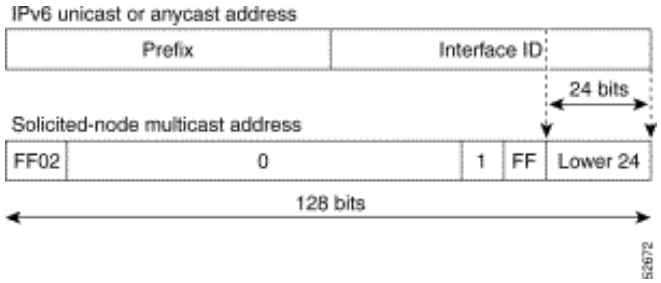
- All-nodes multicast group FF02:0:0:0:0:0:0:1 (scope is link-local);
- Solicited-node multicast group FF02:0:0:0:0:1:FF00:0000/104 for each of its assigned unicast and anycast addresses.

IPv6 routers must also join the all-routers multicast group FF02:0:0:0:0:0:0:2 (scope is link-local).

The solicited-node multicast address is a multicast group that corresponds to an IPv6 unicast or anycast address. IPv6 nodes must join the associated solicited-node multicast group for every unicast and anycast address to which it is assigned. The IPv6 solicited-node multicast address has the prefix FF02:0:0:0:0:1:FF00:0000/104 concatenated with the 24 low-order bits of a corresponding IPv6 unicast or anycast address (see Figure 2). For example, the solicited-node

multicast address corresponding to the IPv6 address 2037::01:800:200E:8C6C is FF02::1:FF0E:8C6C. Solicited-node addresses are used in neighbour solicitation messages.

Figure 2: IPv6 Solicited-Node Multicast Address Format



Note: There are no broadcast addresses in IPv6. IPv6 multicast addresses are used instead of broadcast addresses.

In addition to the lifetime flag described above, RFC 3306 adds a flag specifying that an address is based on a network prefix. It describes how one based on a unicast prefix can generate multicast addresses that are unique to that prefix. One can e.g. for any /64 unicast prefix, use 32 bits to specify different groups, all unique to that prefix. SSM addresses and embedded-RP addresses are also prefix based addresses, but embedded-RP adds yet another flag.

9.1.3 IPv6 Multicast Routing Implementation

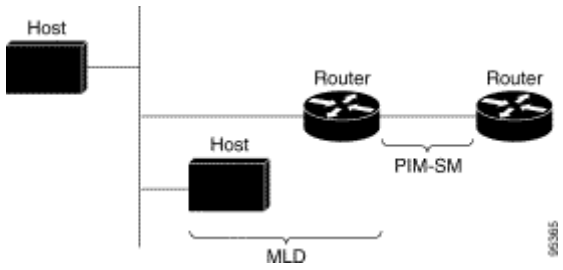
Multicast routing is represented by the tree building procedures and protocols necessary to perform them. On the network side it is currently always PIM (Protocol Independent Multicast). The user signals interest in certain multicast data to the network using MLD (Multicast Listener Discovery) protocol:

- MLD for IPv6. MLD is used by IPv6 routers to discover multicast listeners (nodes that want to receive multicast packets destined for specific multicast addresses) on directly attached links. There are two versions of MLD: MLD version 1 is based on version 2 of the IGMP for IPv4, and MLD version 2 is based on version 3 of the Internet Group Management Protocol (IGMP) for IPv4. IPv6 multicast for Cisco IOS software uses both MLD version 2 and MLD version 1.

- PIM-SM is used between routers so that they can track which multicast packets to forward to each other and to their directly connected LANs.

- PIM in Source Specific Multicast (PIM-SSM) is a subset of PIM-SM functions with the ability to report interest in receiving packets from specific source addresses to an IP multicast address.

Figure 3: IPv6 Multicast Routing Protocols Supported for IPv6



9.1.3.1 Multicast Listener Discovery Protocol for IPv6

To start implementing multicasting in the campus network, users must first define who receives the multicast. The MLD protocol is used by IPv6 routers to discover the presence of multicast listeners (for example, nodes that want to receive multicast packets) on their directly attached links, and to discover specifically which multicast addresses are of interest to those neighbouring nodes. It is used for discovering local group and source-specific group membership. The MLD protocol provides a means to automatically control and limit the flow of multicast traffic throughout the network with the use of special multicast queriers and hosts.

The difference between multicast queriers and hosts is as follows:

- A querier is a network device, such as a router, that sends query messages to discover which network devices are members of a given multicast group.
- A host is a receiver, including routers, that send report messages to inform the querier of a host membership.

A set of queriers and hosts that receive multicast data streams from the same source is called a multicast group. Queriers and hosts use MLD reports to join and leave multicast groups and to begin receiving group traffic.

MLD uses the Internet Control Message Protocol (ICMP) to carry its messages. All MLD messages are link-local with a hop limit of 1, and they all have the router alert option set. The router alert option implies an implementation of the hop-by-hop option header.

MLD has three types of messages:

- Query— General, group-specific, and multicast-address-specific. In a query message, the multicast address field is set to 0 when MLD sends a general query. The general query learns which multicast addresses have listeners on an attached link. Group-specific and multicast-address-specific queries are the same. A group address is a multicast address.
- Report— In a report message, the multicast address field is the specific IPv6 multicast address which the sender is listening.
- Done— In a done message, the multicast address field is the specific IPv6 multicast address which the source of the MLD message is no longer listening.

An MLD report must be sent with a valid IPv6 link-local source address, or the unspecified address (::), if the sending interface has not yet acquired a valid link-local address. Sending reports with the

unspecified address is allowed to support the use of IPv6 multicast in the Neighbour Discovery Protocol.

For stateless auto configuration, a node is required to join several IPv6 multicast groups in order to perform Duplicate Address Detection (DAD). Prior to DAD, the only address the reporting node has for the sending interface is a tentative one, which cannot be used for communication. Therefore, the unspecified address must be used.

MLD provides support for source filtering. Source filtering allows a node to report interest in listening to packets only from specific source addresses (as required to support SSM), or from all addresses except specific source addresses sent to a particular multicast address.

When a host using MLD version 1 sends a leave message, the router needs to send query messages to reconfirm that this host was the last MLD version 1 host joined to the group before it can stop forwarding traffic. This function takes about 2 seconds. This so-called "leave latency" is also present in IGMP version 2 for IPv4 multicast.

9.1.3.2 Protocol Independent Multicast

PIM is an efficient IPv4 and IPv6 routing protocol that is "independent" of any unicast routing table to perform RPF (Reverse Path Forwarding) check. In other words, regardless of which unicast routing protocols are being used in the network to populate the unicast routing table, PIM leverages the existing unicast table content to perform the RPF check function instead of building and maintaining its own separate multicast route table like other protocols – MOSPF or DVMRP.

You can configure IPv6 multicast to use either PIM-SM or PIM-SSM operation, or you can use both PIM-SM and PIM-SSM together in the network.

PIM-Sparse Mode

IPv6 multicast provides support for intradomain multicast routing using PIM-SM. PIM-SM is used in a multicast network when relatively few routers are involved in each multicast and these routers do not forward multicast packets for a group, unless there is an explicit request for the traffic.

Requests for data are accomplished via PIM joins, which are sent hop by hop toward the root node of the tree. The root node of a tree in PIM-SM is the RP in the case of a shared tree or the first-hop router that is directly connected to the multicast source in the case of a shortest path tree (SPT). The RP keeps track of multicast groups and the hosts that are sending multicast packets are registered with the RP by that host's first-hop router.

As a PIM join travels up the tree, routers along the path set up multicast forwarding state so that the requested multicast traffic will be forwarded back down the tree. When multicast traffic is no longer needed, a router sends a PIM prune up the tree toward the root node to prune (or remove) the unnecessary traffic. As this PIM prune travels hop by hop up the tree, each router updates its forwarding state appropriately. Ultimately, the forwarding state associated with a multicast group or source is removed.

A multicast data sender sends data destined for a multicast group. The designated router (DR) of the sender takes those data packets, unicast-encapsulates them, and sends them directly to the RP. The RP receives these encapsulated data packets, de-encapsulates them, and forwards them onto the shared tree. The packets then follow the (*, G) multicast tree state in the routers on the RP tree, being replicated wherever the RP tree branches, and eventually reaching all the receivers for that

multicast group. The process of encapsulating data packets to the RP is called registering, and the encapsulation packets are called PIM register packets.

Designated Router

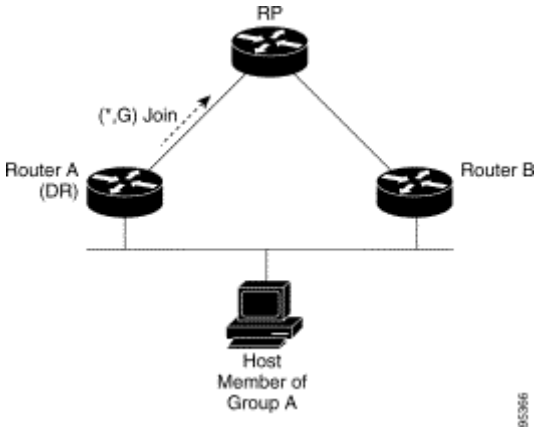
PIM-SM uses the concept of DR when there is more than one router on a LAN segment to forward multicast traffic.

The designated router is responsible for sending PIM register and PIM join and prune messages toward the RP to inform it about host group membership and to forward data on the LAN segment from the upstream routers.

If there are multiple PIM-SM routers on a LAN, a designated router must be elected to avoid duplicating multicast traffic for connected hosts. The PIM router with the highest IPv6 address becomes the DR for the LAN unless you choose to force the DR election using the DR priority option. This allows to specify the DR priority of each router on the LAN segment (default priority = 1) so that the router with the highest priority will be elected as the DR. If all routers on the LAN segment have the same priority, then the highest IPv6 address is again used as the tiebreaker.

Figure 4 illustrates what happens on a multi access segment. Router A and Router B are connected to a common multi access Ethernet segment with Host A as an active receiver for Group A. Only Router A, operating as the DR, sends joins to the RP to construct the shared tree for Group A. If Router B was also permitted to send (*, G) joins to the RP, parallel paths would be created and Host A would receive duplicate multicast traffic. Once Host A begins to source multicast traffic to the group, the DR's responsibility is to send register messages to the RP. If both routers were assigned the responsibility, the RP would receive duplicate multicast packets.

Figure 4: Designated Router Election on a Multi access Segment



If the DR should fail, the PIM-SM provides a way to detect the failure of Router A and elect a failover DR. If the DR (Router A) became inoperable, Router B would detect this situation when its neighbour adjacency with Router A timed out. Because Router B has been hearing MLD membership reports from Host A, it already has MLD state for Group A on this interface and would immediately send a join to the RP when it became the new DR. This step re-establishes traffic flow

down a new branch of the shared tree via Router B. Additionally, if Host A were sourcing traffic, Router B would initiate a new register process immediately after receiving the next multicast packet from Host A. This action would trigger the RP to join the SPT to Host A via a new branch through Router B.

Rendezvous Point

When PIM is configured in sparse mode, you must also choose one or more routers to operate as an RP. An RP is a single common root placed at a chosen point of a shared distribution tree and is configured statically in each box.

PIM DRs forward data from directly connected multicast sources to the RP for distribution down the shared tree. Data is forwarded to the RP in one of two ways:

- Data is encapsulated in register packets and unicast directly to the RP by the first-hop router operating as the DR.
- If the RP has itself joined the source tree, it is multicast-forwarded per the RPF forwarding algorithm described in the "PIM-Sparse Mode" section.

The RP address is used by first-hop routers to send PIM register messages on behalf of a host sending a packet to the group. The RP address is also used by last-hop routers to send PIM join and prune messages to the RP. All routers need to know the RP address.

A PIM router can be an RP for more than one group. Only one RP address can be used at a time within a PIM domain.

Distributing RP information

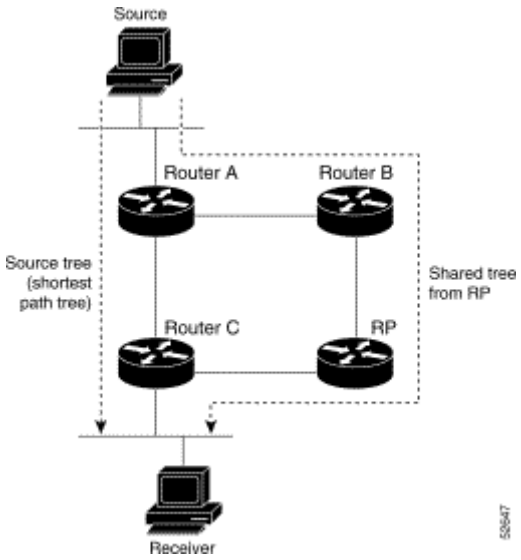
The following mechanisms can be used in IPv6 multicast to distribute the RP information across the whole network:

- Static configuration: The RP can be configured statically on each network device.
- BootStrap Router (BSR): BSR is first specified in RFC2362, the functionality of it is enhanced in the new IETF draft to cover the possibility for administrative scoping in PIM domains using BSR.
- Embedded RP: Embedded RP is a new IPv6 specific mechanism of learning the RP information (e.g. the group to RP mappings) using the encoding of the RP IPv6 address inside of the multicast group destination address. This mechanism removes the need for complicated RP distribution procedures - every router on the multicast data path simply learns the RP automatically from the group address.

PIM Shared Tree and Source Tree (Shortest-Path Tree)

By default, members of a group receive data from senders to the group across a single data distribution tree rooted at the RP. This type of distribution tree is called shared tree or rendezvous point tree (RPT), as illustrated in Figure 5. Data from senders is delivered to the RP for distribution to group members joined to the shared tree.

Figure 5: Shared Tree and Source Tree (Shortest Path Tree)



If the data threshold warrants, leaf routers on the shared tree may initiate a switch to the data distribution tree rooted at the source. This type of distribution tree is called a shortest path tree or source tree. Note that it's not uncommon for the threshold to be zero, which means that the router initiates the switch as soon as it receives the first packet.

The following process describes the move from shared tree to source tree in more detail:

1. Receiver joins a group; leaf Router C sends a join message toward the RP.

2. RP puts the link to Router C in its outgoing interface list.

3. Source sends the data; Router A encapsulates the data in the register and sends it to the RP.

4. RP forwards the data down the shared tree to Router C and sends a join message toward the source. At this point, data may arrive twice at Router C, once encapsulated and once natively.

5. When data arrives natively (un-encapsulated) at the RP, the RP sends a register-stop message to Router A.

6. By default, receipt of the first data packet prompts Router C to send a join message toward the source.

7. When Router C receives data on (S, G), it sends a prune message for the source up the shared tree.

8. RP deletes the link to Router C from the outgoing interface of (S, G).

9. RP triggers a prune message toward the source.

Join and prune messages are sent for sources and RPs. They are sent hop-by-hop and are processed by each PIM router along the path to the source or RP. Register and register-stop messages are not sent hop-by-hop. They are sent by the designated router that is directly connected to a source and are received by the RP for the group.

PIM-Source Specific Multicast

Source specific multicast represents subset of PIM-SM specification which removes all procedures related to the RP and deploys only the source and group specific (S,G) procedures. As an implication, the receiver is required to know his source and group (while full PIM-SM delivers the mechanisms to perform network based source and receivers discovery) and has to signal to the network interest only in that particular (S,G) data.

On the receiver side SSM requires implementation of MLDv2. The network side is required to recognise that it should apply SSM procedures and originate only (S,G) PIM join directly to the source. In the IPv4 case, this recognition is based on the configured SSM address range. IPv6 strictly assigned in its addressing architecture an address range for SSM only and network devices must not apply any RP specific procedures for these addresses.

The deployment of SSM is a fully valid option in both intra and inter domain scenarios.

The deployment scenario with SSM is quite simple - it only requires PIM enabled in the networks and the last hop routers being aware of the SSM address ranges. The drawback is the potential grow of (S,G) forwarding states if the multicast deployment reaches the level of unicast.

The deployment of SSM is still complicated because the lack of support for MLDv2 in the some end user IP stacks (e.g. Windows XP).

Reverse Path Forwarding

Reverse-path forwarding is used for forwarding multicast datagrams. It functions as follows:

- If a router receives a datagram on an interface it uses to send unicast packets to the source, the packet has arrived on the RPF interface.
- If the packet arrives on the RPF interface, a router forwards the packet out the interfaces present in the outgoing interface list of a multicast routing table entry.
- If the packet does not arrive on the RPF interface, the packet is silently discarded to prevent loops.

PIM uses both source trees and RP-rooted shared trees to forward datagrams; the RPF check is performed differently for each, as follows:

- If a PIM router has source-tree state (that is, an (S, G) entry is present in the multicast routing table), the router performs the RPF check against the IPv6 address of the source of the multicast packet.
- If a PIM router has shared-tree state (and no explicit source-tree state), it performs the RPF check on the RP's address (which is known when members join the group).

Sparse-mode PIM uses the RPF lookup function to determine where it needs to send joins and prunes. (S, G) joins (which are source-tree states) are sent toward the source. (*, G) joins (which are shared-tree states) are sent toward the RP.

9.1.4 MRIB

In the IETF definition of Multicast Routing Information Base (MRIB) it is just an unicast routing table containing routes to perform RPF checks and is populated by the unicast routing protocols discussed in the previous chapters.

Cisco implementation used the MRIB term differently (due to the prior implementation before the existence of the new draft) and here it represents the PIM tree database (TIB):

> The (MRIB) is a protocol-independent repository of multicast routing entries instantiated by multicast routing protocols (routing clients). Its main function is to provide independence between routing protocols and the Multicast Forwarding Information Base (MFIB). It also acts as a coordination and communication point among its clients.
>
> Routing clients use the services provided by the MRIB to instantiate routing entries and retrieve changes made to routing entries by other clients. Besides routing clients, MRIB also has forwarding clients (MFIB instances) and special clients such MLD. MFIB retrieves its forwarding entries from MRIB and notifies the MRIB of any events related to packet reception. These notifications can either be explicitly requested by routing clients or spontaneously generated by the MFIB.
>
> Another important function of MRIB is to allow for the coordination of multiple routing clients in establishing multicast connectivity within the same multicast session. MRIB also allows for the coordination between MLD and routing protocols.

There are also other vendors that can populate the MRIB from additional sources. Two common options are static routes and multi protocol BGP.

9.1.5 MFIB

The MFIB is a platform-independent and routing-protocol-independent library for IPv6 software. Its main purpose is to provide an interface with which to read the IPv6 multicast forwarding table and notifications when the forwarding table changes. The information provided by the MFIB has clearly defined forwarding semantics and is designed to make it easy for the platform to translate to its specific hardware or software forwarding mechanisms.

When routing or topology changes occur in the network, the IPv6 routing table is updated, and those changes are reflected in the MFIB. The MFIB maintains next-hop address information based on the information in the IPv6 routing table.

9.1.6 IPv6 Multicast Process Switching and Fast Switching in Cisco Routers

A unified MFIB is used to provide both fast switching and process switching support for PIM-SM and PIM-SSM in IPv6 multicast. In process switching, the route processor must examine, rewrite and forward each packet. The packet is first received and copied into the system memory. The router then looks up the Layer 3 network address in the routing table. The Layer 2 frame is then rewritten with the next-hop destination address and sent to the outgoing interface. The RP also computes the cyclic redundancy check (CRC). This switching method is the least scalable method for switching IPv6 packets.

IPv6 multicast fast switching allows routers to provide better packet forwarding performance than process switching. Information conventionally stored in a route cache is stored in several data

structures for IPv6 multicast switching. The data structures provide optimized lookup for efficient packet forwarding.

In IPv6 multicast forwarding, the first packet is fast-switched if the PIM protocol logic allows it. Also in IPv6 multicast fast switching, the MAC encapsulation header is precomputed. IPv6 multicast fast switching uses the MFIB to make IPv6 destination prefix-based switching decisions. In addition to the MFIB, IPv6 multicast fast switching uses adjacency tables to prepend Layer 2 addressing information. The adjacency table maintains Layer 2 next-hop addresses for all MFIB entries.

The adjacency table is populated as adjacencies are discovered. Each time an adjacency entry is created (such as through the ARP protocol), a link-layer header for that adjacent node is precomputed and stored in the adjacency table. Once a route is determined, it points to a next hop and corresponding adjacency entry. It is subsequently used for encapsulation during switching of packets.

A route might have several paths to a destination prefix, such as when a router is configured for simultaneous load balancing and redundancy. For each resolved path, a pointer is added for the adjacency corresponding to the next-hop interface for that path. This mechanism is used for load balancing across several paths.

9.1.7 Scoped Address Architecture

IPv6 includes support for global and non global addresses. This section describes the usage of IPv6 addresses of different scopes.

A scope zone, or a simply a zone, is a connected region of topology of a given scope. For example, the set of links connected by routers within a particular site, and the interfaces attached to those links, comprise a single zone of site-local scope.

A zone is a particular instance of a topological region (for example, Zone1's site or Zone2's site), whereas a scope is the size of a topological region (for example, a site or a link). The zone to which a particular non global address pertains is not encoded in the address itself, but rather is determined by context, such as the interface from which it is sent or received. Therefore, addresses of a given non global scope may be reused in different zones of that scope. For example, Zone1's site and Zone2's site may each contain a node with site-local address FEC0::1.

Zones of the different scopes are instantiated as follows:

- Each link, and the interfaces attached to him, comprises a single zone of link-local scope (for both unicast and multicast).
- There is a single zone of global scope (for both unicast and multicast), comprising all the links and interfaces in the Internet.
- The boundaries of zones of scope other than interface-local, link-local, and global must be defined and configured by network administrators. A site boundary serves as such for both unicast and multicast.

Zone boundaries are relatively static features and do not change in response to short-term changes in topology. Therefore, the requirement that the topology within a zone be "connected" is intended to include links and interfaces that may be connected only occasionally. For example, a residential node or network that obtains Internet access by dialup to an employer's site may be treated as part

of the employer's site-local zone even when the dialup link is disconnected. Similarly, a failure of a router, interface, or link that causes a zone to become partitioned does not split that zone into multiple zones; rather, the different partitions are still considered to belong to the same zone.

Zones have the following additional properties:

- Zone boundaries cut through nodes, not links (the global zone has no boundary, and the boundary of an interface-local zone encloses just a single interface).
- Zones of the same scope cannot overlap; that is, they can have no links or interfaces in common.
- A zone of a given scope (less than global) falls completely within zones of larger scope; that is, a smaller scope zone cannot include more topology than any larger scope zone with which it shares any links or interfaces.
- Each interface belongs to exactly one zone of each possible scope.

9.2 Cisco IOS IPv6 Multicast

For the minimum required IOS version for multicst features please refer to:

http://www.cisco.com/univercd/cc/td/doc/product/software/ios123/123cgcr/ipv6_c/ftipv6s.htm#wp10 04964

Note 1: On C12000, IPv6 multicast is supported in IOS version 12.0.26S.

Note 2: IPv6 multicast for Cisco IOS software uses MLD version 2. This version of MLD is fully backward-compatible with MLD version 1 (described in RFC 2710). Hosts that support only MLD version 1 will interoperate with a router running MLD version 2. Mixed LANs with both MLD version 1 and MLD version 2 hosts are likewise supported. In a situation with mixed LANs, all nodes default to MLD version 1.

Note 3: IPv6 multicast is supported only over IPv4 tunnels in Cisco IOS Release 12.3(2)T.

9.2.1 Enabling IPv6 Multicast Routing

This task explains how to enable IPv6 multicast routing on all interfaces and to enable multicast forwarding for PIM and MLD on all enabled interfaces of the router.

An IPv6 address must be configured on an interface for the interface to forward IPv6 traffic. Configuring a site-local or global IPv6 address on an interface automatically configures a link-local address and activates IPv6 for that interface. Additionally, the configured interface automatically joins the following required multicast groups for that link:

- Solicited-node multicast group FF02:0:0:0:0:1:FF00::/104 for each unicast and anycast address assigned to the interface.
- All-nodes link-local multicast group FF02::1.

- All-routers link-local multicast group FF02::2.

The following command enables multicast routing on all IPv6-enabled interfaces and enables PIM and MLD on all enabled interfaces of the router:

```
Router(config)# ipv6 multicast-routing
```

9.2.2 Populating the Routing Table for RPF Check

Cisco IOS PIM can use all IPv6 unicast routing protocols in order to perform RPF check - refer to the unicast routing protocol chapters for details on the configuration and availability of the protocols. Similarly to unicast routing the RPF check needs to select one route when several same prefixes are available in several routing protocol tables. It uses the following rules to do it:

1) Exclude IPv6 BGP unicast (SAFI=1) routes by default. It can be enabled by "**`ipv6 rpf use-bgp`**" command.

2) Do longest match on the multicast packet source address across all available routing protocols tables in the exactly same way as the lookup for unicast packet forwarding is done. Include also all multicast specific tables (currently only MP-BGP and static multicast routes). There is no configuration option to change this behaviour.

3) Break ties using the administrative distances if there are prefixes of the same length in several protocol tables. If the administrative distance is still same, prefer static routes above MP-BGP and MP-BGP above unicast routing protocols tables.

On top of the ordinary unicast routing protocols, IOS allows the usage of MP-BGP and static multicast routes for RPF check. The tables created by these two options are not available for unicast packet routing and can therefore be used to create non-congruent unicast and multicast topologies on one physical infrastructure.

Static multicast routes

The IPv4 unicast static route and its multicast alternative "**`ip mroute`**" has been merged in IPv6 into one command:

```
Router(config)# ipv6 route  <prefix> <next-hop> <admin-distance> \\
                                                        unicast/multicast
```

By default administrative distance is set to zero and the route is used for both unicast forwarding and multicast RPF check. As soon as one of the unicast or multicast keywords is entered, it excludes the use of the route for the other option.

MP-BGP deployment

The BGP configuration of a core part of the network is practically identical to the unicast or IPv4 case. Some new BGP features (like peer-group templates or dynamic update groups) might not be available for the IPv6 address families and need to be checked on a case-by-case basis.

Internal BGP core

The iBGP core requires the usual full mesh of peerings or any of the route-reflection or confederation scenarios common in general BGP.

Global BGP process and peer definition

Each BGP speaker has to be configured explicitly with a 32 bit entity (sometimes called IPv4 address) to acquire the BGP router ID necessary in the BGP OPEN message (RFC1771) in order to close the BGP session.

```
router bgp 6680
      bgp router-id 62.40.98.142
      neighbor 6NET_INTERNAL_6NET_PEER peer-group
neighbor 6NET_INTERNAL_6NET_PEER password <removed>
      neighbor 6NET_INTERNAL_6NET_PEER update-source Loopback0
neighbor 2001:798:10::1 remote-as 6680
      neighbor 2001:798:10::1 peer-group 6NET_INTERNAL_6NET_PEER
      neighbor 2001:798:12::1 remote-as 6680
      neighbor 2001:798:12::1 peer-group 6NET_INTERNAL_6NET_PEER
neighbor 2001:798:16::1 remote-as 6680
      neighbor 2001:798:16::1 peer-group 6NET_INTERNAL_6NET_PEER
      neighbor 2001:798:17::1 remote-as 6680
      neighbor 2001:798:17::1 peer-group 6NET_INTERNAL_6NET_PEER
      neighbor 2001:798:20::1 remote-as 6680
      neighbor 2001:798:20::1 peer-group 6NET_INTERNAL_6NET_PEER
```

Activate peers in address families

```
router bgp 6680
address-family ipv6 multicast
      neighbor 6NET_INTERNAL_6NET_PEER activate
      neighbor 6NET_INTERNAL_6NET_PEER send-community
      neighbor 6NET_INTERNAL_6NET_PEER advertisement-interval 1
      neighbor 2001:798:10::1 peer-group 6NET_INTERNAL_6NET_PEER
      neighbor 2001:798:12::1 peer-group 6NET_INTERNAL_6NET_PEER
      neighbor 2001:798:16::1 peer-group 6NET_INTERNAL_6NET_PEER
      neighbor 2001:798:17::1 peer-group 6NET_INTERNAL_6NET_PEER
```

```
      neighbor 2001:798:20::1 peer-group 6NET_INTERNAL_6NET_PEER
exit-address-family
 !
 address-family ipv6
      neighbor 6NET_INTERNAL_6NET_PEER activate
      neighbor 6NET_INTERNAL_6NET_PEER send-community
      neighbor 6NET_INTERNAL_6NET_PEER advertisement-interval 1
      neighbor 2001:798:10::1 peer-group 6NET_INTERNAL_6NET_PEER
      neighbor 2001:798:12::1 peer-group 6NET_INTERNAL_6NET_PEER
neighbor 2001:798:16::1 peer-group 6NET_INTERNAL_6NET_PEER
      neighbor 2001:798:17::1 peer-group 6NET_INTERNAL_6NET_PEER
      neighbor 2001:798:20::1 peer-group 6NET_INTERNAL_6NET_PEER
      no synchronization
 exit-address-family
```

External peerings

Global peers definition

```
      router bgp 6680
      neighbor 6NET_EXTERNAL_NREN_PEER peer-group
            neighbor 2001:718:0:B000::1 remote-as 2852
            neighbor 2001:718:0:B000::1 peer-group 6NET_EXTERNAL_NREN_PEER
            neighbor 2001:718:0:B000::1 password <removed>
            neighbor 2001:798:14:200::2 remote-as 680
            neighbor 2001:798:14:200::2 peer-group 6NET_EXTERNAL_NREN_PEER
      neighbor 2001:798:14:200::2 password <removed>
```

Activate peers in address families

```
      router bgp 6680
      address-family ipv6 multicast
            neighbor 6NET_EXTERNAL_NREN_PEER activate
            neighbor 6NET_EXTERNAL_NREN_PEER send-community
            neighbor 6NET_EXTERNAL_NREN_PEER advertisement-interval 5
            neighbor 2001:718:0:B000::1  peer-group 6NET_EXTERNAL_NREN_PEER

      exit-address-family
      !
      address-family ipv6
      neighbor 6NET_EXTERNAL_NREN_PEER activate
```

```
        neighbor 6NET_EXTERNAL_NREN_PEER send-community
        neighbor 6NET_EXTERNAL_NREN_PEER advertisement-interval 5
        neighbor 6NET_EXTERNAL_NREN_PEER soft-reconfiguration inbound
        neighbor 6NET_EXTERNAL_NREN_PEER route-map FROM-NREN in
        neighbor 2001:718:0:B000::1 peer-group 6NET_EXTERNAL_NREN_PEER
    neighbor 2001:798:14:200::2 peer-group 6NET_EXTERNAL_NREN_PEER
    exit-address-family
```

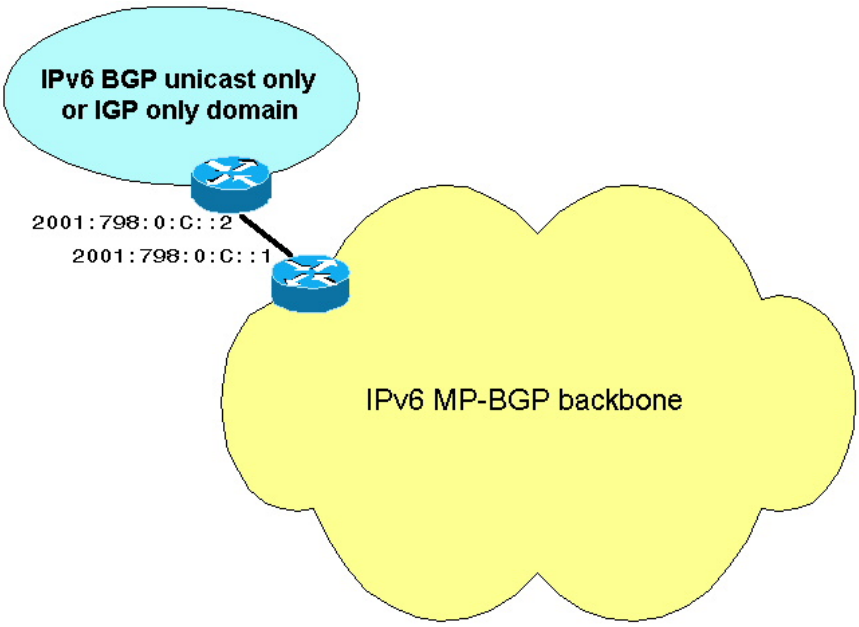
Connecting non-MP-BGP domains

This example covers two cases of backbone clients (connected networks):

1) the client runs only Internal Gateway Protocol (IGP) like IS-IS, RIP or OSPF;

2) the client runs BGP but does not support MP-BGP.

Figure 6 Non-MP-BGP backbone clients



In the first case the backbone border router needs to run the client's IGP on the link between the two and redistribute it into its MP-BGP on behalf of the client - see example in the “Redistribute IGP” section below.

In the second case, the client already originates his BGP prefixes, but only with SAFI=1 (unicast address family). The backbone border router needs to take these prefixes and not only propagate them with SAFI=1 but also inject them with SAFI=2 while keeping all other BGP attributes like AS-path. This case is covered by the example in Translate updates from non-MP-BGP peer section - both keywords "unicast multicast" must be present in the translate-update command in order to propagate updates with both SAFI=1 and SAFI=2.

Redistribute IGP

The BGP neighbours specified in the multicast address family are internal MP-BGP peer towards the backbone routers.

RIP process "test" (or any other user specified process name or routing protocol) needs to be enabled on the interface leading to the client and redistributed into the multicast address family.

```
address-family ipv6 multicast
      neighbor 2001:798:14::2 activate
      neighbor 2001:798:14::3 activate
      redistribute rip test
exit-address-family
```

Translate updates from non-MP-BGP peer

In this case the client can run BGP, but only unicast address family. In the example below the border router has same internal BGP peers activated for both unicast and multicast address family as in the previous example. The `2001:798:0:C::2` peer is the unicast only and external peer and needs to be configured under the address family unicast with the translate-update command. To make this configuration effective and inject the updates with multicast address family the client BGP peer has to be activated under the multicast address family as well although it does not support it.

```
address-family ipv6
      neighbor 2001:798:0:C::2 activate
neighbor 2001:798:0:C::2 translate-update ipv6 multicast unicast
neighbor 2001:798:14::2 activate
neighbor 2001:798:14::3 activate
exit-address-family
!
address-family ipv6 multicast
neighbor 2001:798:0:C::2 activate
neighbor 2001:798:14::2 activate
neighbor 2001:798:14::3 activate
exit-address-family
```

9.2.3 Configuring PIM

The following task configures PIM-SM and PIM-SSM parameters in the global mode or on specific interfaces. By default PIM is enabled on all IPv6 enabled interfaces.

This command is used to configure when a PIM leaf router joins the SPT for the specified groups:

```
Router(config)# ipv6 pim spt-threshold infinity \\
                                          [group-list <access-list-name>]
```

To configure the Rendevous Point for a particular group range use the following command:

```
Router(config)# ipv6 pim rp-address <rp-address> \\
                                                    [group-access-list]
```

If one enters the command

```
Router(config-if)# no ipv6 pim
```

within the interface configuration context **pim** is turned of on this interface.

Configures the DR priority on a PIM router as follows:

```
Router(config-if)# ipv6 pim dr-priority <priority value>
```

One can also set the frequency of PIM hello messages on an interface:

```
Router(config-if)# ipv6 pim hello-interval <seconds>
```

9.2.3.1 Example

The following example shows how to configure a router to use PIM SM using 2001::1 as RP. The following example sets the SPT threshold to infinity to prevent switchover to the source tree when a source starts sending traffic, and sets the PIM hello interval:

```
Router(config)# ipv6 multicast-routing
Router(config)# ipv6 pim rp-address 2001::1
Router(config)# ipv6 pim spt-threshold infinity
Router(config)# interface ethernet 0/0
Router(config-if)# ipv6 pim hello-interval 60
```

9.2.4 Configuring MFIB

Multicast forwarding is automatically enabled when IPv6 multicast routing is enabled. However, a user may want to disable multicast forwarding on the router and its interfaces. To disable IPv6 multicast forwarding once and for all on the whole router enter the following configuration in the general configuration context:

```
Router(config)# no ipv6 mfib
```

If multicast forwarding should only be enabled on some interfaces but not on others, disable multicast on the non-multicast participating interfaces by entering their configuration contexts and typing the following command:

```
Router(config-if)# no ipv6 mfib fast
```

The command disables MFIB interrupt-level IPv6 multicast forwarding of outgoing packets on the specified interface.

9.2.5 Configuring the MLD Protocol

Use the following tasks to customize MLD on each interface, if desired. If multicast is enabled MLD will also automatically be switched on, on every interface. Use the following command to disable it on a specified interface.

```
Router(config-if)# no ipv6 mld router
```

If MLD is switched on the following command configures MLD reporting for a specified group and source.

```
Router(config-if)# ipv6 mld join-group <multicast address of \\
                        group to join> exclude <ipv6 ip of host to exclude>
```

Different MLD timers may be manually configured:

- The maximum response time advertised in MLD queries:

  ```
  Router(config-if)# ipv6 mld query-max-response-time <1-32 seconds>
  ```

- The timeout value before the router takes over as the querier for the interface:

  ```
  Router(config-if)# ipv6 mld query-timeout <60-300 seconds>
  ```

- The frequency at which the Cisco IOS software sends MLD host-query messages:

  ```
  Router(config-if)# ipv6 mld query-interval 90
  ```

  Changing this the query interval should only be performed with caution as it may severely impact multicast forwarding.

9.2.5.1 Example

The following example shows how to configure the query maximum response time, the query timeout, and the query interval on FastEthernet interface 1/0:

```
Router>enable
Router# config terminal
Router(config)# interface FastEthernet 1/0
Router(config-if)# ipv6 mld query-max-response-time 15
Router(config-if)# ipv6 mld query-timeout 30
Router(config-if)# ipv6 mld query-interval 90
```

9.2.6 Resetting the MRIB Connection

The PIM topology table may be cleared. Doing so resets the MRIB connection:

```
Router# clear ipv6 pim topology [group-name | group-address] [reset]
```

9.2.6.1 Example

The following example shows how to clear the PIM topology table and reset the MRIB connection:

```
Router> enable
Router# clear ipv6 pim topology reset
```

9.2.7 Resetting MFIB Traffic Counters

The following command resets all active MFIB traffic counters.

```
Router# clear ipv6 mfib counters [group-name | group-address \\
                                  [source-address | source-name]]
```

9.2.8 Verifying Basic Multicast Configuration and Operation

Various **show** commands can be used to verify the configuration and operation of IPv6 multicast. This chapter will present some of them along with some sample output to give the reader an idea on what is possible.

9.2.8.1 "show ipv6 mfib"

Command Syntax:

```
router# show ipv6 mfib [group-name [source-address]] [verbose]
```

The following example displays the forwarding entries and interfaces in the MFIB. The router is configured for fast switching, and it has a receiver joined to `FF03::1` on Ethernet1/1 and a source (`2001::1:1:20`) sending on Ethernet1/2.

```
Router# show ipv6 mfib
          IP Multicast Forwarding Information Base
Entry Flags: C - Directly Connected, S - Signal, IA - Inherit A flag,
             AR - Activity Required, D - Drop
Forwarding Counts: Pkt Count/Pkts per second/Avg Pkt Size/Kbits per second
Other counts: Total/RPF failed/Other drops
Interface Flags: A - Accept, F - Forward, NS - Negate Signalling
             IC - Internal Copy, NP - Not platform switched
             SP - Signal Present
Interface Counts: FS Pkt Count/PS Pkt Count
(*,FF00::/8) Flags: C
   Forwarding: 0/0/0/0, Other: 0/0/0
   Tunnel0 Flags: NS
(*,FF00::/15) Flags: D
   Forwarding: 0/0/0/0, Other: 0/0/0
(*,FF03::1) Flags: C
   Forwarding: 2/0/100/0, Other: 0/0/0
   Tunnel0 Flags: A NS
```

```
    Ethernet1/1 Flags: F NS
      Pkts: 0/2
  (2001::1:1:200,FF03::1) Flags:
    Forwarding: 5/0/100/0, Other: 0/0/0
    Ethernet1/2 Flags: A
    Ethernet1/1 Flags: F NS
      Pkts: 3/2
  (*,FF10::/15) Flags: D
    Forwarding: 0/0/0/0, Other: 0/0/0
```

9.2.8.2 "show ipv6 mfib active"

This command displays the rate at which active sources are sending to multicast groups.

Command Syntax:

```
router# show ipv6 mfib [group-name | group-address] active [kbps]
```

The following example displays statistics on the rate at which active IP multicast sources are sending information. The router in particular is switching traffic from `2001::1:1:200` to `FF03::1`:

```
Router# show ipv6 mfib active
Active IPv6 Multicast Sources - sending >= 4 kbps
Group: FF03::1
  Source: 2001::1:1:200
    Rate: 20 pps/16 kbps(1sec), 0 kbps(last 128 sec)
```

9.2.8.3 "show ipv6 mfib count "

The command displays summary traffic statistics from the MFIB about the group and source.

Command Syntax:

```
router# show ipv6 mfib [group-name | group-address \\
                                    [source-name | source-address]] count
```

Sample output from a router, which is switching traffic from `2001::1:1:200` to `FF03::1`:

```
Router# show ipv6 mfib count
IP Multicast Statistics
54 routes, 7 groups, 0.14 average sources per group
Forwarding Counts: Pkt Count/Pkts per second/Avg Pkt Size/Kilobits per sec
ond
Other counts: Total/RPF failed/Other drops(OIF-null, rate-limit etc)
Group: FF00::/8
  RP-tree:    Forwarding: 0/0/0/0, Other: 0/0/0
```

```
Group: FF00::/15
  RP-tree:    Forwarding: 0/0/0/0, Other: 0/0/0
Group: FF03::1
  RP-tree:    Forwarding: 2/0/100/0, Other: 0/0/0
  Source: 2001::1:1:200,   Forwarding: 367/10/100/7, Other: 0/0/0
  Tot. shown: Source count: 1, pkt count: 369
Group: FF10::/15
  RP-tree:    Forwarding: 0/0/0/0, Other: 0/0/0
Group: FF20::/15
  RP-tree:    Forwarding: 0/0/0/0, Other: 0/0/0
```

9.2.8.4 "show ipv6 mfib interface"

This command displays information about IPv6 multicast-enabled interfaces and their forwarding status. The router in the example is configured for fast switching:

```
Router# show ipv6 mfib interface
IPv6 Multicast Forwarding (MFIB) status:
    Configuration Status: enabled
    Operational Status: running

MFIB interface       status    CEF-based output
                               [configured,available]
Ethernet1/1          up     [yes       ,yes      ]
Ethernet1/2          up     [yes       ,?        ]
Tunnel0              up     [yes       ,?        ]
Tunnel1              up     [yes       ,?        ]
```

9.2.8.5 "show ipv6 mfib summary"

This command displays summary information about the number of IPv6 MFIB entries and interfaces.

Sample output:

```
Router# show ipv6 mfib summary
IPv6 MFIB summary:
  54     total entries [1 (S,G), 7 (*,G), 46 (*,G/m)]
  33     total MFIB interfaces
```

9.2.8.6 "show ipv6 mld groups"

This command displays the multicast groups that are directly connected to the router and that were learned through MLD.

Command Syntax:

```
router# show ipv6 mld groups [group-name | group-address] \\
                                        [interface-type interface-number]
```

In the following example, the **show ipv6 mld groups** command is used to verify all of the groups joined by FastEthernet interface 2/1, including link-local groups used by network protocols:

```
Router# show ipv6 mld groups FastEthernet 2/1
MLD Connected Group Membership
Group Address          Interface           Uptime       Expires
FF02::2                FastEthernet2/1     3d18h        never
FF02::D                FastEthernet2/1     3d18h        never
FF02::16               FastEthernet2/1     3d18h        never
FF02::1:FF00:1         FastEthernet2/1     3d18h        00:00:27
FF02::1:FF00:79        FastEthernet2/1     3d18h        never
FF02::1:FF23:83C2      FastEthernet2/1     3d18h        00:00:22
FF02::1:FFAF:2C39      FastEthernet2/1     3d18h        never
FF06:7777::1           FastEthernet2/1     3d18h        00:00:26
```

9.2.8.7 "show ipv6 mld groups summary"

The command displays the number of (*, G) and (S, G) membership reports present in the MLD cache.

Sample output:

```
Router# show ipv6 mld groups summary
MLD Route Summary
  No. of (*,G) routes = 0
  No. of (S,G) routes = 0
```

9.2.8.8 "show ipv6 mld interface"

This command displays multicast related information about an interface.

Command Syntax:

```
router# show ipv6 mld interface [type number]
```

Sample output:

```
Router# show ipv6 mld interface FastEthernet 2/1
FastEthernet2/1 is up, line protocol is up
  Internet address is FE80::205:5FFF:FEAF:2C39/10
  MLD is enabled in interface
  Current MLD version is 1
MLD query interval is 10 seconds
```

```
MLD querier timeout is 25 seconds
MLD max query response time is 10 seconds
 Last member query response interval is 1 seconds
 MLD activity: 25 joins, 17 leaves
MLD querying router is FE80::205:5FFF:FEAF:2C39 (this system)
```

9.2.8.9 "show ipv6 mrib client"

This command displays information about the clients of the Multicast Routing Information Base (MRIB).

Command Syntax:

```
router# show ipv6 mrib client [filter] \\
                                [name {client-name | client-name:client-id}]
```

Sample output:

```
Router# show ipv6 mrib client
IP MRIB client-connections
igmp:145        (connection id 0)
pim:146 (connection id 1)
mfib ipv6:3     (connection id 2)
slot 3  mfib ipv6 rp agent:16   (connection id 3)
slot 1  mfib ipv6 rp agent:16   (connection id 4)
slot 0  mfib ipv6 rp agent:16   (connection id 5)
slot 4  mfib ipv6 rp agent:16   (connection id 6)
slot 2  mfib ipv6 rp agent:16   (connection id 7)
```

9.2.8.10 "show ipv6 mrib route"

This command displays MRIB route information.

Command Syntax:

```
router# show ipv6 mrib route [source-address | source-name | *] \\
                        [group-name | group-address [prefix-length]] [summary]
```

Sample output:

```
Router# show ipv6 mrib route
IP Multicast Routing Information Base
Entry flags:L - Domain-Local Source, E - External Source to the Domain,
    C - Directly-Connected Check, S - Signal, IA - Inherit Accept, D -
 Drop
Interface flags:F - Forward, A - Accept, IC - Internal Copy,
```

```
    NS - Negate Signal, DP - Don't Preserve, SP - Signal Present,
    II - Internal Interest, ID - Internal Disinterest, LI -
 Local Interest,
    LD - Local Disinterest


(*,FF00::/8) RPF nbr:2001:0DB8:6::6 Flags:L C
  Tunnel5 Flags:NS


(*,FF00::/15) Flags:D
.
.
.
(2001:0DB8:999::99, FF07::1) RPF nbr:2001:0DB8:999::99 Flags:
  POS1/0 Flags:A NS
  POS4/0 Flags:F NS
```

9.2.8.11 "show ipv6 mroute"

Using the `show ipv6 mroute` command is a good way to dynamically verify that multicast IPv6 data is flowing.

Command Syntax:

```
router# show ipv6 mroute [group-name | group-address \\
                          [source-address | source-name]] [summary] [count]
```

The following is a sample output from the `show ipv6 mroute` command:

```
Router# show ipv6 mroute ff07::1
Multicast Routing Table
Flags:D - Dense, S - Sparse, B - Bidir Group, s - SSM Group,
       C - Connected, L - Local, I - Received Source Specific Host Report,
       P - Pruned, R - RP-bit set, F - Register flag, T - SPT-bit set,
       J - Join SPT
Timers:Uptime/Expires
Interface state:Interface, State

(*, FF07::1), 00:04:45/00:02:47, RP 6:6:6::6, flags:S
  Incoming interface:Tunnel5
  RPF nbr:6:6:6::6
  Outgoing interface list:
    POS4/0, Forward, 00:04:45/00:02:47
```

```
(999:999:999::99, FF07::1), 00:02:06/00:01:23, flags:SFT
  Incoming interface:POS1/0
  RPF nbr:999:999:999::99
  Outgoing interface list:
    POS4/0, Forward, 00:02:06/00:03:27
```

9.2.8.12 "show ipv6 mroute active"

This command displays the active multicast streams on the router.

Command Syntax:

```
router# show ipv6 mroute [group-name | group-address] active [kbps]
```

Sample output:

```
Router# show ipv6 mroute active
Active IPv6 Multicast Sources - sending >= 4 kbps
Group:FF05::1
 Source:2001::1:1:1
   Rate:11 pps/8 kbps(1sec), 8 kbps(last 8 sec)
```

9.2.8.13 "show ipv6 pim group-map"

The following command may be used to view the IPv6 multicast group mapping table.

Command Syntax:

```
router# show ipv6 pim group-map [group-name | group-address]
```

Sample output:

```
Router# show ipv6 pim group-map
FF33::/32*
    RP      :::
    Protocol:SSM
    Client  :config
    Groups  :0
    Info    :
.
.
.
FF34::/32*
    RP      :::
    Protocol:SSM
    Client  :config
```

```
    Groups  :0
    Info    :
```

9.2.8.14 "show ipv6 pim interface"

To view the configuration pertaining to PIM on an interface use this command.

Command Syntax:

```
router# show ipv6 pim interface [type number]
```

Sample output:

```
Router# show ipv6 pim interface
Interface          PIM  Nbr   Hello  DR
                        Count Intvl  Prior

Ethernet0          on   0     30     1
    Address:FE80::208:20FF:FE08:D7FF
    DR     :this system
POS1/0             on   0     30     1
    Address:FE80::208:20FF:FE08:D554
    DR     :this system
POS4/0             on   1     3600   1
    Address:FE80::208:20FF:FE08:D554
    DR     :FE80::250:E2FF:FE8B:4C80
POS4/1             on   0     3600   1
    Address:FE80::208:20FF:FE08:D554
    DR     :this system
Loopback0          on   0     30     1
    Address:FE80::208:20FF:FE08:D554
    DR     :this system
```

9.2.8.15 "show ipv6 pim neighbour"

This command is used to display the PIM neighbors discovered by the Cisco IOS software.

Command Syntax:

```
router# show ipv6 pim neighbor [interface-type interface-number | count]
```

Sample output:

```
Router# show ipv6 pim neighbor
Neighbor Address             Interface         Uptime        Expires DR  P
ri  Bidir
```

```
FE80::202:FCFF:FE3C:A438     FastEthernet2/0   3d18h         00:01:28    1
    B
```

9.2.8.16 "show ipv6 pim range-list"

This command shows information about IPv6 multicast range lists.

Command Syntax:

```
router# show ipv6 pim range-list [config] [rp-address | rp-name]
```

Sample output:

```
Router# show ipv6 pim range-list
 FF33::/32 Up:00:42:01
 FF34::/32 Up:00:42:01
 FF35::/32 Up:00:42:01
 FF36::/32 Up:00:42:01
 FF37::/32 Up:00:42:01
 FF38::/32 Up:00:42:01
 FF39::/32 Up:00:42:01
 FF3A::/32 Up:00:42:01
 FF3B::/32 Up:00:42:01
 FF3C::/32 Up:00:42:01
 FF3D::/32 Up:00:42:01
 FF3E::/32 Up:00:42:01
 FF3F::/32 Up:00:42:01
```

9.2.8.17 "show ipv6 pim topology"

Use the following command to display PIM topology table information for a specific group or all groups.

Command Syntax:

```
router# show ipv6 pim topology [summary] [group-name | group-address] \\
                                                [source-address | source-name]
```

Sample output from the **show ipv6 pim topology** command using the **route-count** keyword:

```
Router# show ipv6 pim topology route-count
PIM TT Summary
 No. of group ranges = 47
 No. of (*,G) routes = 29
 No. of (S,G) routes = 1
```

```
No. of (S,G)RPT routes = 0
```

9.2.8.18 "show ipv6 pim tunnel"

The following command displays information about the PIM register encapsulation and de-encapsulation tunnels on an interface.

Command Syntax:

```
router# show ipv6 pim tunnel [interface-type interface-number]
```

Sample output from the **show ipv6 pim tunnel** command on an RP:

```
Router#show ipv6 pim tunnel
Tunnel0*
 Type  :PIM Encap
 RP    :100::1
 Source:100::1
Tunnel0*
 Type  :PIM Decap
 RP    :100::1
 Source: -
```

Sample output from the **show ipv6 pim tunnel** command on a non-RP:

```
Router#show ipv6 pim tunnel
Tunnel0*
 Type  :PIM Encap
 RP    :100::1
 Source:2001::1:1:1
```

9.2.9 Troubleshooting IPv6 Multicast

Use **debug** commands to help you troubleshoot an IPv6 multicast environment. This task describes the commands to display debugging information on IPv6 multicast.

9.2.9.1 "debug ipv6 mfib"

The following command enables debugging output on the IPv6 MFIB.

Command Syntax:

```
router# debug ipv6 mfib [adjacency | signal | db | init | mrib | pak | \\
                                                    ps] [group-name | group-address]
```

- **adjacency:** Keyword to output only information about adjacency management activity.

- **signal:** If used MFIB data-driven signaling to routing protocols will be displayed.
- **db:** This keyword pertains to route database management activity.
- **init:** Show initialization or de-initialization activity.
- **mrib:** Use this command to show communication with the MRIB.
- **pak:** If used all packet forwarding activity will be displayed.
- **ps:** This displays process-level-only additional packet forwarding activity.
- `group-name | group-address`: Optional parameters to specifiy an IPv6 group address or name to which to restrain the output to.

You must use at least one keyword to enable debugging output for this command.

9.2.9.2 "debug ipv6 mld"

This command enables debugging on PIM protocol activity. It helps discover whether the MLD protocol activities are working correctly. In general, if MLD is not working, the router process never discovers that there is a host on the network that is configured to receive multicast packets.

The messages displayed by the **`debug ipv6 mld`** command show query and report activity received from other routers and hosts. Use this command in conjunction with **`debug ipv6 pim`** to display additional multicast activity, to learn more information about the multicast routing process, or to learn why packets are forwarded out of particular interfaces.

Command Syntax:

```
router# debug ipv6 mld [group-name | group-address | interface-type]
```

In Cisco IOS Release 12.0(26)S, the syntax is as follows:

```
router# debug ipv6 mld [group group-name | group-address | \\
                                                  interface interface-type]
```

- `group-name | group-address`: (Optional) IPv6 address or name of the multicast group.
- `interface-type`: (Optional) Interface type. For more information, use the question mark (**?**) online help function.

9.2.9.3 "debug ipv6 pim explicit"

To display information related to the explicit tracking of hosts, use the **`debug ipv6 mld explicit`** command.

Command Syntax:

```
router# debug ipv6 mld explicit [group-name | group-address]
```

When the optional `group-name` or `group-address` argument is not used, all debugging information is displayed.

Example:

The following example shows how to enable information to be displayed about the explicit tracking of hosts. The command output is self-explanatory:

```
Router# debug ipv6 mld explicit

00:00:56:MLD:ET host FE80::A8BB:CCFF:FE00:800 report for FF05::6 (0 srcs)
on Ethernet1/0
00:00:56:MLD:ET host FE80::A8BB:CCFF:FE00:800 switch to exclude for
FF05::6 on Ethernet1/0
00:00:56:MLD:ET MRIB modify for (*,FF05::6) on Ethernet1/0 new 100, mdf
100
```

9.2.9.4 "debug ipv6 pim"

This command helps discover whether the PIM protocol activities are working correctly.

The messages displayed by the **`debug ipv6 pim`** command show all PIM protocol messages, such as joins and prunes, received from or sent to other routers. Use this command in conjunction with **`debug ipv6 mld`** to display additional multicast activity, to learn more information about the multicast routing process, or to learn why packets are forwarded out of particular interfaces.

Command Syntax:

```
router# debug ipv6 pim [group-name | group-address | interface-type | \\
                                                                neighbor]
```

- `group-name | group-address`: (Optional) IPv6 address or name of the multicast group.
- `interface-type`: (Optional) Interface type. For more information, use the question mark (**`?`**) online help function.
- **`neighbor`**: (Optional) Debug statistics related to hello message processing and neighbor cache management.
- **`bsr`**: (Optional) Debug statistics specific to bootstrap router (BSR) protocol operation.

9.2.9.5 "debug ipv6 mrib client"

The **`debug ipv6 mrib client`** command is used to display the activity in the MRIB associated with clients such as Protocol Independent Multicast (PIM) and Multicast Listener Discovery (MLD). If you are having difficulty with your client connections, use this command to display new clients being added and deleted.

The **debug ipv6 mrib client** command also displays information on when a new client is added to or deleted from the MRIB, when a client connection is established or torn down, when a client binds to a particular MRIB table, and when a client is informed that there are updates to be read.

Command Syntax:

```
router# debug ipv6 mrib client
```

9.2.9.6 "debug ipv6 mrib io"

To enable debugging on Multicast Routing Information Base (MRIB) I/O events, use the **debug ipv6 mrib io** command.

Use this command to display information when clients open and close MRIB I/O connections, when MRIB entry and interface updates are received and processed from clients, and when MRIB entry and interface updates are sent to clients.

Command Syntax:

```
router#  debug ipv6 mrib io
```

This command has no arguments or keywords.

9.2.9.7 "debug ipv6 mrib route"

This command displays update information related to the route database made by MRIB clients, which is then redistributed to the clients.

Use this command to monitor MRIB route activity when discontinuity is found between the MRIB and the client database or between the individual client databases.

Command Syntax:

```
router#  debug ipv6 mrib route [group-name | group-address]
```

- `group-name | group-address`: (Optional) IPv6 address or name of the multicast group.

9.2.9.8 "debug ipv6 mrib table"

Use the **debug ipv6 mrib table** command to display information on new MRIB tables being added and deleted.

Command Syntax:

```
router# debug ipv6 mrib table
```

9.3 JUNIPER implementation

9.3.1 Enabling IPv6 multicast routing

IPv6 multicast routing is enabled when PIM protocol is configured for IPv6. As it is for time being the only multicast routing protocol, there is no need of a special command. It is important to

remember, that Juniper routers requires hardware called Tunnel-PIC to work as PIM-SM RP or DR. Tunnel-PIC is needed to create tunnels and for packet encapsulation.

9.3.2 Configuring PIM

The command `edit protocols pim` enters PIM configuration subtree when being at the root of the JUNOS configuration tree.

```
[edit]
protocols {
    pim {
        dense-groups {
            addresses;
        }
        disable;
        import [ policy-names ];

        interface interface-name {
            disable;
            hello-interval seconds;
            mode (dense | sparse | sparse-dense);
            priority number;
            version version;
        }
        rib-group group-name;
        rp {
            auto-rp (announce | discovery | mapping);
            bootstrap-export [ policy-names ];
            bootstrap-import [ policy-names ];
            bootstrap-priority number;
            local {
                family (inet | inet6) {
                    disable;
                    address address;
                    group-ranges {
                        destination-mask;
                    }
                    hold-time seconds;
                    priority number;
                }
            }
            static {
                address address {
                    version version;
                    group-ranges {
                        destination-mask;
        }
        traceoptions {
            file name <replace> <size size> <files number> <no-stamp>
                <(world-readable | no-world-readable)>;
            flag flag <flag-modifier> <disable>;
        }
    }
}
```

9.3.2.1 RIB configuration for PIM-SM

```
user@router# set rib-group inet6 routing_table_name
```

PIM will use the RIB `routing_table_name` for RPF check. `routing_table_name` must be configured somehow in routing-options configuration tree.

```
[edit routing-options]
rib-groups {
        routing_table_name {
            import-rib inet6.2;
```

9.3.2.2 PIM-SM RP configuration

At the time being, only static RP configuration is possible for IPv6 (including Junos 6.0). The following command makes the JUNIPER router act as an RP:

```
user@router# set rp local family inet6
```

The following command adds other RPs:

```
user@router# set rp static address ipv6_rp_address
```

Entering the RP configuration tree, more options can be given:

- **`group-ranges`** `IPv6_prefix`: Prefixes for which the RP is valid;
- **`version`** `(1..2)`: Sets up PIM version of the RP.

9.3.2.3 PIM interfaces configuration

PIM is enabled only on interfaces declared in:

```
[edit protocols pim interface interface_name]
```

subtree.

To add the settings on every interface, `interface_name` can be set to "all". By default PIM operates in sparse mode on an interface.

It is then possible to disable a specific interface with `set disable` setting in PIM interface configuration subtree.

For every PIM interface, it is possible to configure:

`mode` *`mode`* PIM mode (sparse or dense), Sparse mode is used by default;

`version` *`(1..2)`* PIM version (1 or 2);

`.hello-interval` *`(0..255)`* PIM hello interval;

`.priority` *`(0..4294967295)`* PIM DR priority.

Minimal PIM-SM configuration, when router act as RP is shown bellow. In this example `2001:808:506::1` is used as RP address:

```
pim {
        rp {
            local {
                family inet6 {
                    address 2001:808:506::1;
                }
            }
        }
        interface all {
            mode sparse;
        }
    }
```

9.3.3 Configuring MLD

The JUNOS software supports MLD versions 1and 2, but version 2 is supported for SSM include mode only. MLD is automatically enabled on all broadcast interfaces when you configure PIM but it is possible to edit some MLD parameters in `edit protocols mld` subtree.

The following options can be given:

`query-interval`	When to send host query messages (1..1024 seconds);
`query-last-member-interval`	When to send group query messages (seconds);
`query-response-interval`	How long to wait for a host query response (seconds);
`robust-count`	Expected packet loss on a subnet (2..10).

It is possible to configure MLD for a specific interface:

```
edit interface interface_name
```

`interface_name` can be set to "`all`" to setup every interface.

`disable`	Disable MLD on this interface;
`static`	Static group or source membership;
`version`	Set MLD version number on this interface (1..2).

9.3.4 Verifying basic IPv6 connection, configuration and options

PIM neighbors are displayed by using command:

```
show pim neighbors inet6
```

Sample output of `show pim neighbors inet6` command is shown bellow:

```
Instance: PIM.master

Interface        IP V Mode      Option      Uptime Neighbor addr
ge-0/0/0.0        6 2           HPLGB     00:01:36 fe80::210:7bff:fe81:cb96
ge-0/0/0.0        6 2           HPLGB     00:01:36 fe80::230:b6ff:fe1c:98a8
```

PIM interfaces are displayed by using command:

```
show pim interfaces inet6
```

Sample output of `show pim interfaces inet6` command is shown bellow:

```
Instance: PIM.master

Name               Stat Mode     IP V State Count DR address
ge-0/0/0.0         Up   Sparse    6 2 NotDR     2 fe80::230:b6ff:fe1c:98a8
ge-0/2/0.0         Up   Sparse    6 2 DR        0 fe80::205:85ff:fe26:843e
```

PIM RPs are displayed by using command:

```
show pim rps inet6
```

Sample output of `show pim rps inet6` command is shown bellow:

```
Instance: PIM.master

Family: INET6
RP address      Type     Holdtime  Timeout  Active groups  Group prefixes
2001:808:506::1 static      0        None         1           ff00::/8
```

9.3.5 Troubleshooting IPv6 multicast

Global PIM statistics are accessible with command:

```
show pim statistics
```

Sample output of `show pim statistics` command is shown bellow:

```
PIM Message type       Received        Sent    Rx errors
V2 Hello                     70         517          0
V2 Register                   0           0          0
V2 Register Stop              0           0          0
```

```
V2 Join Prune                0           0           0
V2 Bootstrap                28           0           0
V2 Assert                 1195           0           0
V2 Graft                     0           0           0
V2 Graft Ack                 0           0           0
V2 Candidate RP              0           0           0
V1 Query                     0           0           0
V1 Register                  0           0           0
V1 Register Stop             0           0           0
V1 Join Prune                0           0           0
V1 RP Reachability           0           0           0
V1 Assert                    0           0           0
V1 Graft                     0           0           0
V1 Graft Ack                 0           0           0
AutoRP Announce              0           0           0
AutoRP Mapping               0           0           0
AutoRP Unknown type          0

Global Statistics

Unknown type                                 0
V1 Unknown type                              0
Unknown Version                              0
Neighbor unknown                             0
Bad Length                                   0
Bad Checksum                                 0
Bad Receive If                               0
Rx Intf disabled                             0
Rx V1 Require V2                             0
Rx V2 Require V1                             0
Rx Register not RP                           0
Rx Register no route                         0
Rx Register no decap if                      0
Null Register Timeout                        0
RP Filtered Source                           0
Rx Unknown Reg Stop                          0
Rx Join/Prune no state                       0
Rx Join/Prune on upstream if                 0
Rx sparse join for dense group               0
```

```
Rx Graft/Graft Ack no state              0
Rx Graft on upstream if                  0
Rx CRP not BSR                           0
Rx BSR when BSR                          0
Rx BSR not RPF if                        0
Rx unknown hello opt                    70
Rx data no state                         0
Rx RP no state                           0
Rx aggregate                             0
Rx malformed packet                      0
No RP                                    7
No register encap if                     0
No route upstream                       21
Nexthop Unusable                         0
RP mismatch                              0
RPF neighbor unknown                     0
Rx Joins/Prunes filtered                 0
```

Global multicast statistics are accessible with command:

```
show multicast statistics inet6
```

Multicast statistics are constructed from the forwarding statistics, which are gathered at 30-second intervals. Sample output of `show multicast statistics inet6` command is shown bellow:

```
Address family: INET6
Interface: local
    Routing protocol:                    Mismatch error:                 0
    Mismatch:                       0    Mismatch no route:              0
    Kernel resolve:                 0    Routing notify:                 0
    Resolve no route:               0    Resolve error:                  0
    In kbytes:                      0    In packets:                     0
    Out kbytes:                     0    Out packets:                    0
Interface: ge-0/0/0.0
    Routing protocol:             PIM    Mismatch error:                 0
    Mismatch:                       0    Mismatch no route:              0
    Kernel resolve:                 4    Routing notify:                 0
    Resolve no route:             339    Resolve error:                  0
    In kbytes:                      0    In packets:                     0
    Out kbytes:                   153    Out packets:                 1770
```

```
Interface: ge-0/2/0.0
    Routing protocol:          PIM   Mismatch error:               0
    Mismatch:                    0   Mismatch no route:            0
    Kernel resolve:              3   Routing notify:               0
    Resolve no route:            0   Resolve error:                0
    In kbytes:                 153   In packets:                1770
    Out kbytes:                  0   Out packets:                  0

Resolve requests on interfaces not enabled for multicast: 0
Resolve requests with no route to source: 0
Routing notifications on interfaces not enabled for multicast: 0
Routing notifications with no route to source: 0
Interface mismatches on interfaces not enabled for multicast: 0
Group membership on interfaces not enabled for multicast: 0
```

RPF information can be displayed by using command

```
Router> show multicast rpf inet6 address
```

Sample output of `show multicast rpf inet6` command is shown bellow:

```
multicast RPF table: inet6.0, 10 entries

2001:808:0:4::/64
    Protocol: Direct
    Interface: ge-0/2/0.0

2001:808:0:4::2/128
    Protocol: Local

2001:808:506::/64
    Protocol: Direct
    Interface: ge-0/0/0.0

2001:808:506::1/128
    Protocol: Local

fe80::/64
    Protocol: Direct
    Interface: ge-0/0/0.0
```

```
fe80::205:85ff:fe26:8400/128
    Protocol: Local

fe80::205:85ff:fe26:843e/128
    Protocol: Local

ff02::2/128
    Protocol: PIM

ff02::5/128
    Protocol: OSPF

ff02::d/128
    Protocol: PIM
```

The entries in the multicast forwarding table can be displayed with command:

```
root@router> show multicast route inet6
```

Sample output of **`show multicast route inet6`** command is shown bellow:

```
Family: INET6
Group           Source prefix     Act Pru InIf  NHid  Session Name
ff0b::1234      2001:808:0:4:210:4bff:fe91:9120/128 A   P  70    0
ff0b::1234      2001:808:506:0:220:afff:fe2e:f093/128 A   F  67    335
```

PIM join/prune statistics and informations about PIM groups can be displayed with command

```
root@router> show pim join inet6
```

Sample output of `show pim join inet6` command is shown bellow:

```
Instance: PIM.master Family: INET6

Group: ff0b::1234
    Source: *
    RP: 2001:808:506::1
    Flags: sparse,rptree,wildcard
    Upstream interface: local
```

```
Group: ff0b::1234
    Source: 2001:808:0:4:210:4bff:fe91:9120
    Flags: sparse
    Upstream interface: ge-0/2/0.0

Group: ff0b::1234
    Source: 2001:808:506:0:220:afff:fe2e:f093
    Flags: sparse
    Upstream interface: ge-0/0/0.0
```

PIM source RPF state for a prefix range can be displayed with command:

```
root@router> show pim source inet6 IPv6_prefix
```

Sample output of **`show pim join inet6`** command is shown bellow:

```
Instance: PIM.master Family: INET6

2001:808:0:4:210:4bff:fe91:9120
    Prefix: 2001:808:0:4::/64
    Upstream interface: ge-0/2/0.0
    Upstream neighbor: Direct

2001:808:506::1
    Prefix: 2001:808:506::1/128
    Upstream interface: local
    Upstream neighbor: Local

2001:808:506:0:220:afff:fe2e:f093
    Prefix: 2001:808:506::/64
    Upstream interface: ge-0/0/0.0
    Upstream neighbor: Direct
```

9.4 6WIND implementation

6WIND uses actually pim6sd Kame daemon. The configuration of the dameon can be done in advanced mode only and is exactly the same as the Kame pim6sd dameon (described next).

Nevertheless, there is a 6WIND test image that includes now MP-BGP with the support of IPv6 multicast routes and static multicast routes. The pim6sd daemon was modified to perform RPF check on the MRIB.

Below are some 6OS commands and the corresponding explanations for IPv6 multicast routing setup.

9.4.1 Monitoring

The following command displays the IPv6 Multicast VRF/FIB:

```
sixwind{show-route} show ipv6 mroute [...]
```

This command displays the IPv6 Multicast MP-BGP4+/RIB table:

```
sixwind{show-route} show ipv6 mbgp [...]
```

9.4.2 Configuration

The following command adds an IPv6 multicast static route to the configuration:

```
sixwind{myconf-rtg} mroute X::X/M X::X
```

Use this command to provision the set of IPv6 neighbors that can send IPv6 multicast SAFI:

```
sixwind{myconf-rtg-dynamic-bgp} address-family ipv6 multicast
```

9.5 KAME pim6sd Implementation (and home made add-ons)

9.5.1 Enabling PIM-SM

On a system running the KAME IPv6 implementation PIM-SM can be enabled with the command:

```
# pim6sd [-c configfile] [-d [debug_level [,debug_level]]] [-f]
```

The `-d` option is for debugging only and makes it possible to display some information messages. The `-f` option runs the daemon on foreground.

9.5.2 Configuring PIM-SM

The configuration statements must be written in the configuration file, one per line, followed by a semi-colon. Below is a copy of the man page for *`pim6sd.conf`*, which details every configuration option.

9.5.2.1 "log option"

The "**`log option;`**" directive specifies debug messages to be printed out. Each option usually specifies a subset of the messages to be printed. If an option begins with no, it means that the set of messages that are specified by the option will not be printed. For example, `` `all nomld' `` means that all the messages except MLD related ones will be printed. For other valid options please refer to the man pages.

9.5.2.2 "reverselookup (yes | no)"

The "**reverselookup (yes | no);**" directive specifies if a hostname for an IPv6 address should be resolved on logging. "**yes**" means a hostname should be resolved, and "**no**" means it should not. By default, a hostname is not resolved.

9.5.2.3 "phyint interface [disable]"

The "**phyint interface [disable];**" directive specifies pim6sd to ignore the interface even if the interface is multicast-capable. Note that PIM will be activated on all interfaces by default. Interfaces are specified in the form of "*name unit*", such as gif0 and eth1.

9.5.2.4 phyint interface [preference preference] [metric metric] [nolistener]

The "**phyint interface [preference preference] [metric metric] [nolistener];**" directive specifies the preference and/or metric values when sending a PIM assert message on the interface. If another optional parameter **nolistener** is specified, pim6sd will not send any MLD packets on the interface. This option is usually meaningless but will be useful when MLD messages are noisy (e.g. when debugging) and there is surely no listener on the interface.

9.5.2.5 "default_source_preference preference"

The "**default_source_preference preference;**" directive specifies a default preference value when sending a PIM assert message. Preferences are used by assert elections to determine upstream routers. Currently pim6sd(8) does not have an effective method to obtain preferences and metrics from the unicast routing protocols, so you may want to set a default value by hand. The default preference is 1024.

9.5.2.6 "default_source_metric metric"

The "**default_source_metric metric;**" directive specifies a default metric value when sending a PIM assert message. Since pim6sd cannot get an effective metric of unicast routing, it is recommended that preferences are set such that metrics are never consulted. However, default metrics may also be set, and their default value is 1024.

9.5.2.7 "granularity second"

The "**granularity second;**" directive specifies a timer granularity in seconds. The default value is 5.

9.5.2.8 "hello_period period coef"

The "**hello_period period coef;**" directive specifies the period in second between 2 hello messages. "coef" is the coefficient to determine the hello holdtime; the holdtime will be period * coef. The default values of the period and the coefficient are 30 and 3.5, respectively. The default holdtime is 105 seconds as a result.

9.5.2.9 "join_prune_period period coef"

The "**`join_prune_period period coef;`**" directive specifies the period in second between 2 join/prune messages. "`coef`" is the coefficient to determine the join/prune hold-time; the holdtime will be `period * coef`. The default values of the period and the coefficient are 60 and 3.5, respectively. Consequently, the default holdtime is 210 seconds.

9.5.2.10 "data_timeout timer"

The "**`data_timeout timer;`**" directive specifies the time after which (S,G) state for a silent source will be deleted. The default value is 210.

9.5.2.11 "register_suppression_timeout interval"

The "**`register_suppression_timeout interval;`**" specifies the interval between receiving a Register-Stop and allowing PIM Register to be sent again. The default value is 60.

9.5.2.12 "probe_time timer"

The "**`probe_time timer;`**" specifies the time between sending a null Register and the Register-Suppression-Timer expiring unless it is restarted by receiving a Register-Stop. The default value is 5.

9.5.2.13 "assert_timeout interval"

The "**`assert_timeout interval;`**" specifies the interval between the last time an Assert is received and the time at which the assert is timeout. The default value is 180.

9.5.2.14 "cand_rp [interface] [time time] [priority priority]"

The "**`cand_rp [interface] [time time] [priority priority];`**" directive specifies to act as a Candidate Rendezvous Point(RP). It is recommended to specify `cand_rp` only in typical usage. All other parameters are optional and will be set automatically. If an interface is specified, pim6sd will search for a global address on the specified interface and set the address in Candidate RP Advertisements. An optional parameter time specifies the interval of two succeeding advertisements in seconds. Its default value is 60. 2.5 * `time` will be set to Candidate-RP- Advertisement messages. Another optional parameter priority specifies the priority of the RP. The default value is 0, which means the highest priority.

9.5.2.15 "group_prefix prefix"

When acting as a Rendezvous Point, "**`group_prefix prefix;`**" specifies a group prefix that the RP will handle.

9.5.2.16 "cand_bootstrap_router [interface] [time time] [priority priority] [masklen masklen]"

The "**`cand_bootstrap_router [interface] [time time] [priority priority] [masklen masklen];`**" directive specifies to act as a Candidate Bootstrap Router. It is recommended to specify **`cand_bootstrap_router`** only in typical usage. All other parameters are optional and will be set automatically. If an interface is specified, pim6sd(8) will search for a global address on the specified interface and set the address in Bootstrap messages. An optional parameter time specifies the interval of two succeeding bootstraps in seconds. Its default value is 60. Another optional parameter priority specifies the priority of the BSR. The default value is 0, which means the lowest priority. Hash mask length can also be specified by the masklen parameter. Its value, masklen, must be no less than 0 and no greater than 128.

9.5.2.17 "switch_register_threshold rate rate interval interval"

The "**`switch_register_threshold rate rate interval interval;`**" directive specifies the threshold that a RP switches to a shortest path tree, which is valid only when acting as an RP. `rate` specifies the threshold in bits per second, and `interval` specifies the interval of checking the rate in seconds. The default values are 50000 and 20, respectively.

9.5.2.18 "switch_data_threshold rate rate interval interval"

The "**`switch_register_threshold rate rate interval interval;`**" directive specifies the threshold that a last hop router switches to a shortest path tree. `rate` specifies the threshold in bits per second, and `interval` specifies the interval of checking the rate in seconds. The default values are 50000 and 20, respectively.

9.5.3 PIM information and troubleshooting

The following command are detailed in this section:

```
pim6stat
netstat -g
netstat -gs
```

9.5.3.1 pim6stat

`pim6stat` displays the following IPv6 multicast information:

Multicast Interface Table

It enumerates every interface of the system (with addresses) and shows for any of them the multicast interface number and the corresponding status:

`DISABLED` if the interface is disabled in PIM configuration

`PIM` if there is a neighbor on the interface

`NO-NBR` if there is no PIM neighbor on the link

`QRY` if the system is the MLD querier on the link

`DR` if the system is the PIM DR on the link

PIM Neighbor List

For every interface, the neighbor addresses (link-local and global addresses) and timers are displayed.

MLD Querier List

For every interface, the querier link-local address is displayed with the corresponding timers.

MRIB

Every (S,G) and (*,G) flows are listed, with:

- Interfaces were PIM Join messages were received and corresponding expiration timer;
- Interfaces were PIM Prune messages were received;
- Interfaces were MLD Leave messages were received;
- Asserted interfaces;
- Incoming interfaces;
- Outgoing interfaces.

RP Set

In this section, there is the following information:

- BSR address with priority and timeout;
- RP addresses with for each of them:
  - Next hop link-local address;
  - Group prefixes;
  - Priority;
  - RP Holdtime;
  - RP age timer.

9.5.3.2 "netstat –g"

The command "`netstat -g`" displays the following information:

IPv6 Multicast Interface Table

It gives for every interface the corresponding multicast interface identifier, the number of packets coming in and going out. If a rate is applied on an interface, it is also displayed in this section.

IPv6 Multicast Forwarding Cache

All multicast active flows are listed in this section, with corresponding source and group addresses, the number of packets transmitted and the interfaces where the packets are forwarded.

9.5.3.3 ´"netstat –gs"

The command "`netstat -gs`" displays multicast forwarding global statistics (e.g. multicast forwarding cache lookups, datagrams received on the wrong interface…).

9.6 Single Domain Multicast Scenarios (Cisco)

Single domain scenarios represent the cases, where the whole network is under control of one administration.

9.6.1 Congruent-native topology

In this simplest possible case, where whole network is fully IPv6 and multicast enabled, no special configuration is required, the unicast routing will fully take care of any RPF check requirements:

1) Enable globally multicast routing:

```
Router(config)# ipv6 multicast-routing
```

This enables PIM and MLD on all interfaces enabled for IPv6.

2) Decide for the RP distribution method;

a) configure statically all routers:

```
router(config)# ipv6 access-list range1
router(config-ipv6-acl)# permit ipv6 any FF0B::/16
router(config-ipv6-acl)# permit ipv6 any FF1B::/16
router(config-ipv6-acl)# permit ipv6 any FF3B::/16
router(config-ipv6-acl)# exit

router(config)# ipv6 access-list v4gw
router(config-ipv6-acl)# permit ipv6 any \\
                                        FF3E:30:2001:700:1:FFFF::/96
router(config-ipv6-acl)# exit

router(config)# ipv6 pim rp-address 2001:610:14:5145::145 range1
router(config)# ipv6 pim rp-address 2001:660:10A:6802::1
router(config)# ipv6 pim rp-address 2001:700:E000:501::2 v4gw
```

b) deploy only SSM - this does not require any configuration, SSM `FF3x::/32` range of multicast group addresses must be in use only.

c) deploy Embedded RP - no configuration necessary across the network, Embedded RP address range (`FF70::/12` + embedded RP address) must be in use only. The RP itself needs to be configured as follows:

```
router(config)# ipv6 access-list RP-embedded
router(config-ipv6-acl)# permit ipv6 any FF7B:140:2001:620:0:FE::/96
router(config-ipv6-acl)# exit

router(config)# ipv6 pim rp-address 2001:620:0:FE::1 RP-embedded
```

In this case `2001:620:0:FE::1` is the local router address.

d) BSR

Most vendors have some support for this. This is a dynamic protocol where one router is elected to send bootstrap information to routers in the BSR domain. The BSR domain is typically all routers within an organizational domain, but it need not be. Some routers are configured as candidate RPs, they will all send advertisements to the BSR router. The BSR router will elect which RPs should be used, and they will be included in the BSR messages.

9.6.2 Non-congruent topology

Non-congruent topology here means that the IPv6 multicast topology is different from IPv6 unicast topology.

This can happen when equipments are not IPv6 multicast enable or when administrators prefer to use dedicated equipments or link for IPv6 multicast for some administrative reason.

9.6.2.1 Tunneling

When it is not possible to have a native link for IPv6 multicast, tunneling can be used. (IPv6 multicast over IPv6 unicast or IPv6 multicast over IPv4 unicast). The advantage of the first tunneling technique is that it makes IPv6 multicast completely independant of IPv4. The following lines recalls how to setup IPv6/IPv4 tunnels and IPv6/IPv6 tunnels:

1) IPv6 in IPv4 tunnel:

```
interface Tunnel10
 description tunnel to other-site
 no ip address
 ipv6 address 2002::2/64
 ipv6 enable
```

```
 tunnel source 192.168.15.2
 tunnel destination 192.168.15.1
 tunnel mode ipv6ip
```

2) IPv6 in IPv6 tunnel

```
interface Tunnel11
 no ip address
 ipv6 address 2001:ABCD:6000::2/64
 tunnel source 2001:798:C::1
 tunnel destination 2001:ABCD:2000::2
 tunnel mode ipv6
```

9.6.2.2 Routing

There is no IGP to exchange IPv6 multicast prefixes. Moreover, inside a domain, BGP is not always deployed (eg. inside an end-site) and it not possible to exchange IPv6 multicast prefixes using "ipv6 multicast" address family of MP-BGP. This raises the problem of multicast routing inside the domains where MP-BGP is not available:

1) Separate multicast routers are available (see Figure 7) - this case allows to create the RPF table using some dynamic routing protocol.

Care must be taken, that the protocols running on unicast only routers and multicast only routers do not interfere and do not exchange any routing information.

The multicast router must only have routing information to the tunnel end points (/128 IPv6 or /32 IPv4 prefixes) or a static default route.

The tunnel end points must never be resolved over the tunnel interface itself but through some physical interface.

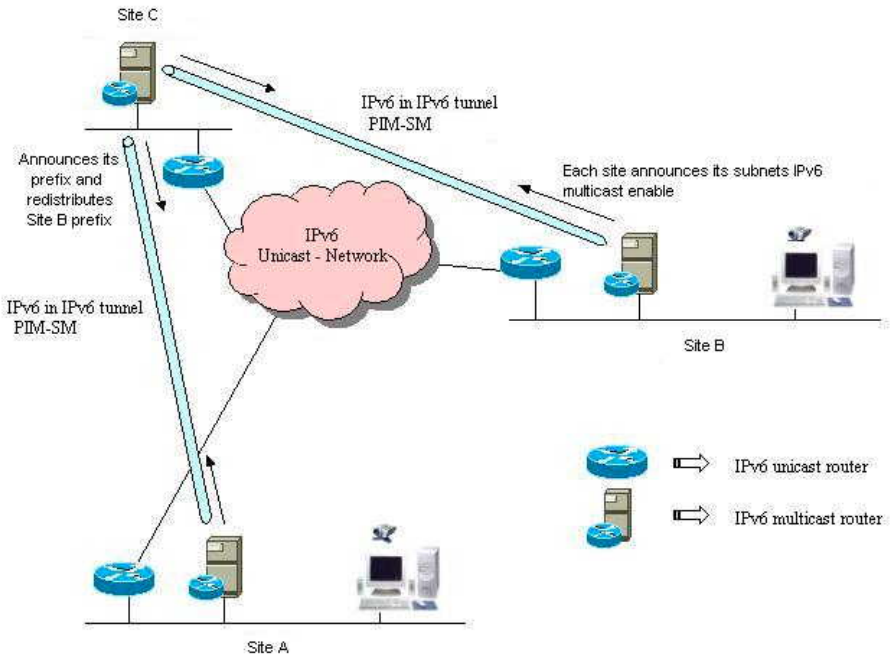


Figure 7: separate routers case

The example below shows how to enable RIPng protocol on the IPv6 multicast only router of the site A in the figure above.

Enable routing process (RIPng) globally:

```
router(config)# ipv6 router rip test
```

Enable interfaces for RIP:

```
router(config)# interface Tunnel10
router(config-if)# ipv6 rip test enable
router(config-if)# exit

router(config)# interface Ethernet0/0
router(config-if)# ipv6 rip test enable
```

If the routing protocol allows, the Ethernet interface should be made passive or a routing updates filter list should be applied to deny all incoming routes and avoid any interaction with the unicast router of site A.

2) The access router on each site has to run both unicast and multicast - no possibility to put the two functions on two physical boxes. In that case the only feasible possibility (another option would be

MP-BGP but it is not typically run till the end-sites) to forward unicast and multicast in their respective links is static routing for RPF checks.

Example for site A:

It is possible to setup 2 different multicast static routes:

```
router(config)# ipv6 route  <prefix_site_B> <next_hop_address> multicast
router(config)# ipv6 route  <prefix_site_C> <next_hop_address> multicast
```

As site 1 is an end-leaf, it is also possible to setup a multicast static default route:

```
router(config)# ipv6 route ::/0 <next_hop_address> multicast
```

In this example, `<next_hop_address>` is the IPv6 address of router C that is configured inside the tunnel.

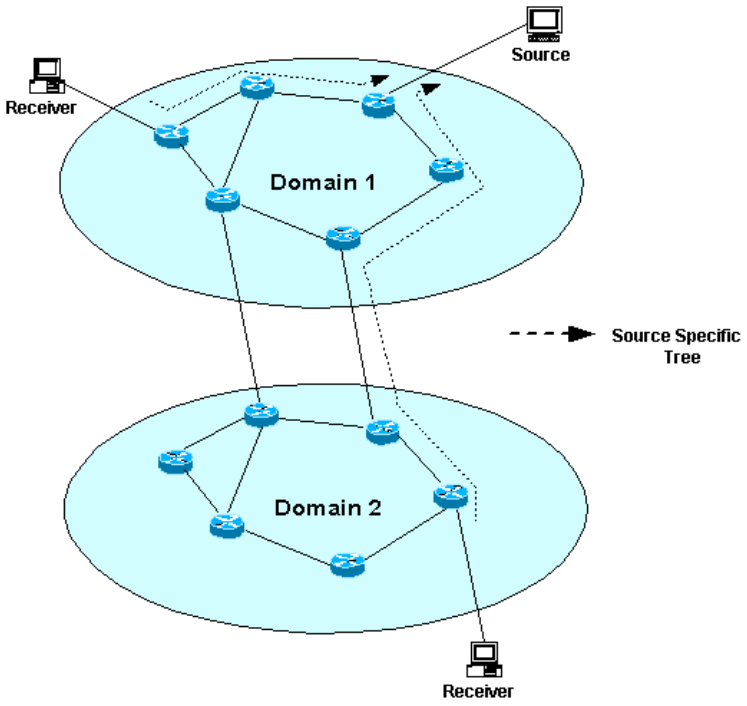
9.7 RP distribution method

Exactly same methods for the RP distribution are available as discussed in the section 9.6.

9.8 Source Specific Multicast

As already mentioned, the deployment of SSM is a fully valid and simplest option in both Intra and Inter domain scenarios. Nevertheless, there is today a lack of implementations of MLDv2 on hosts. This results that there is almost no practice with SSM so far.

Figure 8 SSM example of operation in the Interdomain Scenario



9.9 Interdomain Multicast Scenarios

Interdomain scenarios represent the cases, where the network is divided into several administrative domains based only on the equipment ownership and not related to any protocol functionality.

Interdomain multicast in IPv4 allowed to each domain to administer its own RP via the definition of Multicast Source Discovery Protocol (MSDP) developed to exchange information about sources between the domains. This allowed each RP to issue PIM (S,G) joins to sources in other domains when having receivers for group G.

The operation of the protocol has proven to be troublesome and not scalable for many applications and IETF therefore so far strictly refused to discuss this protocol for IPv6.

When speaking about Interdomain in IPv6 the administrative domains are therefore forced to share the RP unless other mechanisms are used. In the Internet environment this administrative domain is typically also a BGP Autonomous System and BGP will be in most cases the routing protocol run between the domains.

The IETF discussion about Interdomain multicast in IPv6 now concentrates around Source Specific Multicast, which could replace Any Source Multicast (ASM in the sense of the full RFC2362 implementation) using multiple one-to-many trees in order to create full mesh of any-to-any communications.

9.9.1 MP-BGP Deployment

It needs to be emphasized that MP-BGP is not "a MUST" when the network administrator decides to run multicast. If the topology used to deliver unicast and multicast is congruent (or basically exactly same) then there is no need to enable MP-BGP. Congruent in the terms of BGP means:

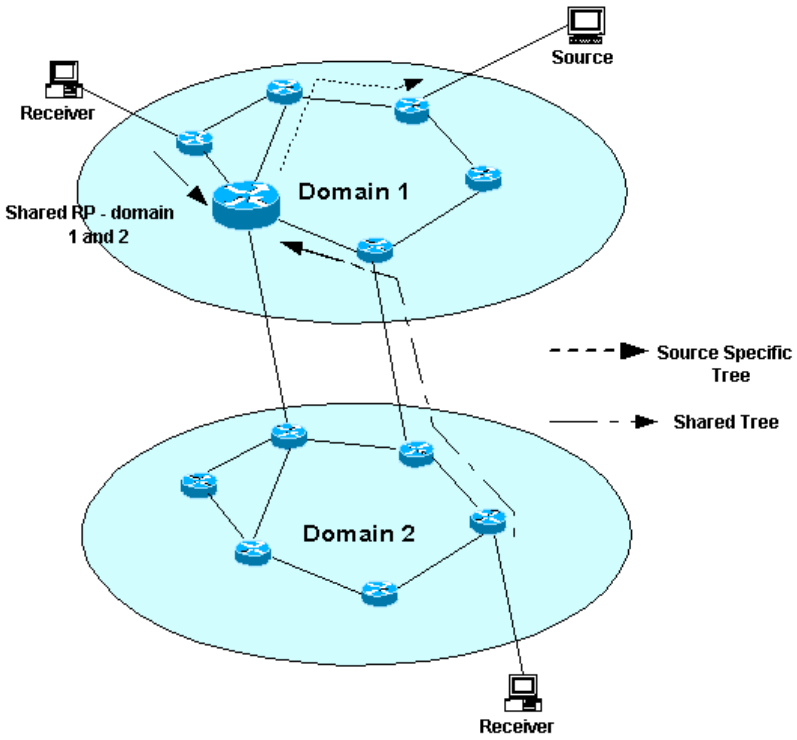
1) all BGP next hops are equal for all potential unicast and multicast BGP routes;

2) the unicast and multicast BGP tables would contain exactly same prefixes and with same BGP attributes.

9.9.2 Any Source Multicast

Any Source Multicast in the IPv6 case requires all PIM domains to share one RP for particular multicast group range, which has many technical and administrative problems. At the moment there is a lack of experience and best practices in this area.

Figure 9 ASM example of operation in the Interdomain scenario



9.9.3 Embedded RendezVous Point

The procedures of Embedded RP can be used in both intra and inter domain scenarios but the main idea of it is to resolve the necessity of the RP group to mappings distribution between domains. As opposed to SSM, all the network devices all the way must implement Embedded RP.

9.9.4 Shared static RP

In this scenario, all PIM domains of the inter-network need to be configured with at least one common RP to handle groups, which will have sources and receivers in all domains. Each domain can have its own set of RPs to handle group ranges specific for each of the local domains. Potentially, the static configuration can be used for the interdomain shared RP and in intradomain it can be substituted by a dynamic distribution of the RP information (BSR, Embedded).

The static configuration represents certain administration burden but if the IP addressing is planned sensibly, the shared RP can be moved to different physical locations inside of the particular hosting domain without actually changing the IP address of it. The shared RP should be physically positioned in the backbone (if the whole inter-network has some hierarchy) so all the users of it have approximately same physical distance to it.

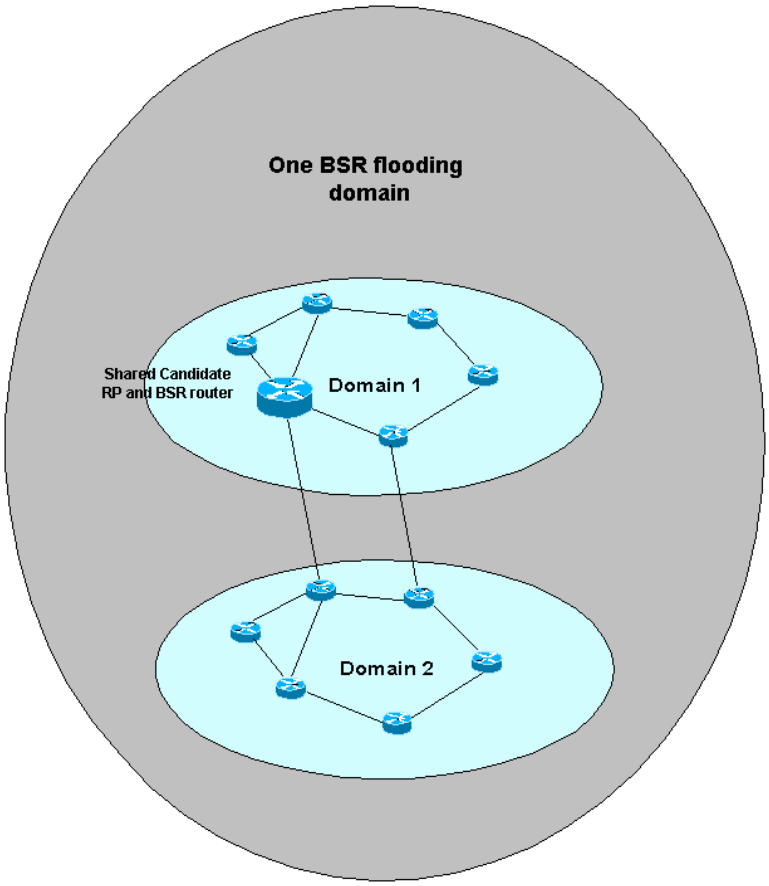
Static configuration has also the advantage of avoiding debugging of pure PIM control information distribution on top of multicast data distribution.

Shared static RP can be a good solution when the inter-network remains small. It is not realistic in the global interdomain network.

9.9.5 Shared BSR domain

In this scenario the whole inter-network running multicast represents just one BSR domain - the identical BSR information is flooded freely without any constrains.

Figure 10 Shared BSR flooding domain example



All the domains must therefore share the exactly same set of RPs for all groups. The distribution of locally significant RP information would not be possible using BSR but only via static configuration. The implementation of the new BSR draft might bring the possibility to use administrative scoping for BSR messages in this scenario to help to solve the problem above.

This represents highly cooperative and hierarchical scenario, one entity manages the operation of the whole inter-network with BSR router and candidate RPs most probably placed in the interconnecting backbone.

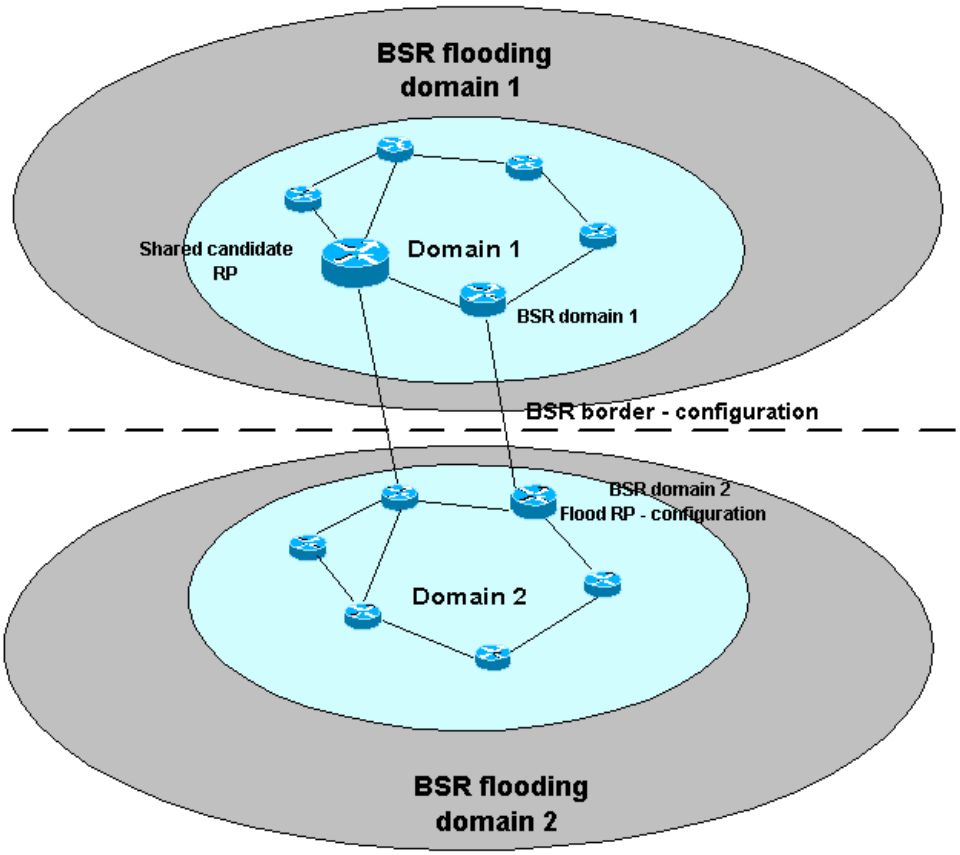
The distribution of the shared RP information is administratively flexible, but the debugging is more complicated - RP distribution and data distribution needs to be debugged.

Distributing RP information via BSR is a good solution in a small network (company, university…) It can't be used on several administrative domains as it not possible to controle anouncements. The new scoped BSR might improve the situation as it offers the possibility to create a BSR hierarchy and controle anouncements in every domain.

9.9.6 Constrained BSR domains

In this scenario each PIM domain would also be a separate BSR domain. Each domain entirely blocks any BSR flooding/traffic to any other domain. The domains need to agree on a common set of RPs and group ranges, which will be flooded identically in each domain. The BSR routers need to allow a static configuration of RPs, which need to be advertised (e.g. without any BSR messages from the candidate RPs to the BSR routers). On top of those common RPs each domain can flood its own information about local RP inside of the domain.

Figure 11 Constrained BSR flooding domains example



This scenario requires again certain level of administrative coordination, but decreases the size of the BSR domains and allows to constrain debugging of BSR flooding to each of the domains with each domain responsible for the distribution inside of the domain. It gives also each domain full flexibility to advertise locally significant RPs internally via BSR.

10 Implementing DNS for IPv6

If one is already familiar with nameservice for IPv4, running it for IPv6 addresses is not much different. In principle there is just a new type of resource record and a new notation for storing resource records for reverse address resolution.

10.1 Basic Operation

10.1.1 New record types

To store IPv6 addresses in DNS, a new type of record was introduced in RFC 2874 [RFC2874]. While IPv4 addresses are stored in the resource record (RR) "*A*", IPv6 adresses uses the new RR "*AAAA*".

Speaking about reverse address resolution the reverse IPv6 address is stored in a PTR record similar to IPv4, but a different format (nibble format) and a different top level zone (`ip6.arpa.`) are used.

10.1.2 What nameserver should I use?

This introduction only discusses the widespread BIND nameserver. Nevertheless, BIND exists in several major versions, 4, 8 and 9. Generally speaking, regardless which major version is used, one should always deploy the newest minor version. This is important, because in the newest versions imminent security holes are fixed.

Nameservice is a key service in the internet, and one should always keep it as secure as possible.

At present the most recent versions are BIND 4.9.11, BIND 8.3.7/8.4.5 and BIND 9.2.4/9.3.0. If A server is running an older version it should be upgraded. Sometimes there are issues why one still needs to run a BIND 8 nameserver (e. g. performance or scalability). BIND4 is deprecated and we strongly recommend upgrading to a newer main release.

10.1.2.1 BIND 4

In case you are still running a BIND 4 nameserver, you definitely are one of the bad boys in the global nameservice community. BIND 4 is old, deprecated, insecure, uses questionable mechanisms and hasn't been maintained for a long time now. An urgent advise is to upgrade at least to version 8. Every distribution of BIND 8 contains detailed information how to migrate from version 4 and scripts reformatting config files. Please use it. There is almost no good reason why to stick to version 4 today.

Due to this, BIND 4 configuration is not covered in this guide. Another one is that, due to the deprecation of BIND 4, we never tested it and never prepared a suitable configuration file (which has a different format than newer BIND versions).

10.1.2.2 BIND 8

If you are running BIND 8 (hopefully the newest minor version), you are on the lucky side and can instantly use it to store IPv6 resource records. BIND 8 knows about *AAAA* records and can answer such reqests. Therefore, for a quick start, BIND 8 is sufficient. If you are running BIND 8.4.x you

can even use IPv6 addresses in your configuration files, as it is able send and receive IPv6 packets (in contrast to BIND 8.3.x). Therefore, if you like you can use BIND 8.4.x in an IPv6-only environment.

10.1.2.3 BIND 9

BIND 9 is the first choice for the IPv6 expert. If you decide to store your IPv6 resource records in a delegated subdomain on a second nameserver you really should use BIND 9 for that server. BIND 9 can store *AAAA* resource records and -- with proper configuration -- can use IPv6 traffic.

10.1.3 Storing an IPv6 addresses (AAAA Record) in the nameservice

Generally speaking, an *AAAA* entry is just an additional entry next to the *A* record for the name of an interface. But it can be dangerous to simply add an *AAAA* record for each present name in a domain. Many new applications query an *AAAA* record before they fall back to query an *A* record. This way it might happen that an interface seems to be unreachable, if the interface is not configured for IPv6 or if it is not connected to the IPv6 network with proper routing procedures.

To prevent such and other derived errors or to not risk endangering the already running IPv4 nameservice, it is sometimes advisable to store all IPv6 records in a subdomain for the beginning. It is common to use a subdomain like `ipv6.yourdomainname.org` for this.

Overall there is nothing to say against storing an *A* and an *AAAA* record for a given name at the same time. But if you do that, you have to be perfectly sure the host you are addressing is really reachable over IPv6. We would advocate this method, but we made the experience, that in doing so the most errors and the most misunderstandings happened. So for the beginning, an `"ipv6"` subdomain is very helpful.

10.1.3.1 IPv6 records in a main zonefile

If you want to place IPv6 entries in your subdomain `ipv6.yourdomainname.org` you can simply add them to your main zonefile. An example with the domain '`uni-muenster.de`':

```
;$ORIGIN uni-muenster.de. ; this line is added for clarification and ; as a reminder. All entires in this file ; belong to the zone uni-muenster.de

atlan IN A 128.176.191.83

atlan.ipv6 IN AAAA 3ffe:400:10:100:201:2ff:feb5:3806
```

The `$ORIGIN` entry is obsolete here, the zone usually is predefined in the `named.conf` configuration file. Like in this example an `$ORIGIN` entry is often added to enhance human readability of the zonefile definitions.

10.1.3.2 IPv6 records in an include file

Often zonefiles are generated with a script, e.g. the names are extracted from a database and the file is assembled automatically. If one does not want to enhance or to change these scripts, it is helpful to add IPv6 entries in a seperate file which is included in the main zone with an `$INCLUDE` statement. An example configuration may look like this:

Zonefile db.uni-muenster.de:

```
;$ORIGIN uni-muenster.de.
```

```
atlan IN A 128.176.191.83
;
$INCLUDE "db.include.ipv6.uni-muenster.de" ipv6.uni-muenster.de.
; or
; $INCLUDE "db.include.ipv6.uni-muenster.de" ipv6
The includefile db.include.ipv6.uni-muenster.de would contain:
;$ORIGIN ipv6.uni-muenster.de. ; again just a reminder !!
atlan IN AAAA 3ffe:400:10:100:201:2ff:feb5:3806
```

The first value in the `$INCLUDE` statement defines the filename of the includefile and the second value defines the subdomain entries in this includefile should belong to. Therefore the "`ipv6.`" part can be left out in the includefile.

10.1.3.3 IPv6 records in a second nameserver

The third and possibly most secure method is to store IPv6 entries in second nameserver and to delegate the `ipv6` subdomain to that server. This way the production nameserver remains untouched and is not endangered by tests and reconfigurations of the IPv6 zones and records. An example for the configuration of the delegating nameserver would be:

```
;$ORIGIN uni-muenster.de.
ipv6.uni-muenster.de. IN NS ns.ipv6.uni-muenster.de.
ns.ipv6.uni-muenster.de. IN A 128.176.191.83
```

Now one can run a second nameserver on the host `128.176.191.83`. That nameserver would be responsible exclusively for the zone `ipv6.uni-muenster.de` and would contain only the IPv6 entries and records.

Nevertheless, this second nameserver should be operated like a productive server (appropriate authorization, secondary servers, etc.). If the delegating nameserver is able to send IPv6 packets (like 8.4.x or 9.x), one could also use an *AAAA* entry in the above example.

To answer general questions on how to set up a nameserver and how to delegate a zone to a second nameserver within its zone, you might want to check the book "DNS and BIND" (O'Reilly), which is always helpful for these tasks.

10.1.4 Reverse address resolution

If you want to store a reverse address (to resolve an IPv6 address into an name), things differ a bit to IPv4. For IPv6 reverse resolution the same Resource Record (PTR) is used, but with a different textual representation.

For IPv4, the address is written in reverse order, separated by a dot at the 8-bit boundary and it is delegated within the zone `in-addr.arpa` (e.g. "`176.128.in-addr.arpa.`"). The notation for IPv6 is a bit different, it is also written backwards with dots as separators, but it is separated every four bits (nibble boundary) and the delegation is done within the zone `ip6.arpa`. (e.g.

"0.0.5.0.8.3.6.0.1.0.0.2.ip6.arpa."). This is called nibble format. In the past the delegation for IPv6 addresses was done in the ip6.int. zone, therefore today it might still be common and often necessary to use a delegation like "0.0.5.0.8.3.6.0.1.0.0.2.ip6.int."

With the example of the University of Münster we would like to show what a proper zonefile should look like. The DFN is delegation the zone 0.0.5.0.8.3.6.0.1.0.0.2.ip6.arpa. to the University's nameserver. To store PTR records in the nameservice at first we have to configure the local nameserver as a master for the above zone.

This is done in the named.conf and may look like this:

```
zone "0.0.5.0.8.3.6.0.1.0.0.2.ip6.arpa"{
type master;
file "db.uni-muenster.de.ip6.arpa";
};
```

The file db.uni-muenster.de.ip6.arpa is the zonefile for this zone and contains the *PTR* records like:

```
; $ORIGIN 0.0.5.0.8.3.6.0.1.0.0.2.ip6.arpa.
6.0.8.3.5.b.e.f.f.f.2.0.1.0.2.0.0.0.0.1.0 IN PTR atlan.ipv6.unimuenster.de.
5.f.4.7.8.d.e.f.f.f.8.1.0.e.2.0.0.0.0.2.0 IN PTR lemy.ipv6.unimuenster.de.
```

Again, the *$ORIGIN* entry is only a reminder. Like every name in the nameservice the complete PTR is now assembled from the entry in the zonefile (left part) and the delegated zone (right part, derived from the *$ORIGIN*).

Querying the nameserver for

5.f.4.7.8.d.e.f.f.f.8.1.0.e.2.0.0.0.0.2.0.0.0.5.0.8.3.6.0.1.0.0.2.ip6.arpa.

would now return the name "lemy.ipv6.uni-muenster.de".

It is laborious to type a reverse notation of an IPv6 address by hand. There are various small programs and scripts for many operating systems to help with this task, "*ipv6calc*" or "*host*" for example. E.g."host –n 2001:638:500:200:2e0:18ff:fed8:74f5" will yield the reverse notation mentioned in the last paragraph.

As mentioned before, in the past IPv6, PTR records were delegated in the top level domain ip6.int.. There are still some applications and libaries that try to resolve reverse IPv6 addresses in this domain instead of ip6.arpa.. To be on the safe side it is better to also ask the delegating nameservice administration for delegation of the according domain within ip6.int.. In the named.conf configuration file you would create a similar entry as above, pointing to the same zonefile:

```
zone "0.0.5.0.8.3.6.0.1.0.0.2.IP6.INT"{
type master;
file "db.uni-muenster.de.ip6.arpa";
};
```

As an alternative you can name the zonefile differently (e.g. db.uni-muenster.de.ip6.int), but create a hard link (*ln*) with db.uni-muenster.de.ip6.arpa (*nix systems only).

10.1.5 A6 and DNAME records/binary format (RFC2874)

RFC2874 introduces the new resource records *A6* and *DNAME* and the bitstring format. *A6* and *DNAME* were developed to ease renumbering with the help of chained resource records, where every nameserver only holds that part of the IPv6 address it has under control. Bitstring labels aim for the same purpose and should additionally overcome constraints in the nibble format, where IPv6 addresses could only be separated at the 4-bit boundary.

Without going further into details, please note that these new records and notations were "deprecated". After RFC2874 was published, the internet community raised fears that chained resource records would slow down name lookups, would raise the danger of severe misconfiguration and might even break the nameservice. Due to this, *A6* and *DNAME* records were degraded to experimental in RFC 3363 [RFC3363], which practically means 'deprecated'. Further information why this was necessary is explained in RFC 3364 [RFC3364].

So, for running an IPv6 nameservice today, it is sufficient to concentrate on *AAAA* records and reverse delegation within the `ip6.arpa.` domain and don't waste time thinking about possible methods of IPv6 address name resolution.

10.1.6 ip6.int. vs. ip6.arpa.

Until the middle of 2001, reverse IPv6 addresses were delegated in the top level domain `ip6.int.`. Very early some people voiced their opinion that `.int` stands for 'international' and shouldn't serve for adminstrative purposes in the Internet. One should use the domain `.arpa` for storing reverse addresses.

RFC3152/BCP49 [RFC3152] (published in August 2001) then demanded that in the future, IPv6 address are to be stored in the `ip6.arpa.` zone and no longer in `ip6.int.`. The use of `ip6.int.` for reverse name resolution was unfortunately widespread and common usage in all applications and operating systems, esp. all 6bone addresses (the `3ffe::/16` address space) were located in the `ip6.int.` zone. But when RFC 3152 was published, further usage of `ip6.int.` was forbidden and the RFC didn't cover the problem of the transition to `ip6.arpa.`. It was impossible to switch all nameservers immediately, because it is unrealistic to expect every single internet application to change it's behaviour to switch from `ipv6.int.` to `ip6.arpa.`. During the transition it often was necessary to ask for reverse addresses in both zones, while it was unclear for a long time in which zone to ask first or if both queries return an answer which one should be used.

Aggravatingly, for a long time only the production prefixes (address space `2001::/16`) were delegated in `1.0.0.2.ip6.arpa.`, 6bone prefixes were not. This problem was finally fixed in October 2004 and 6bone addresses can now be found in `e.f.f.3.ip6.arpa`.

When running a nameservice for reverse IPv6 addresses, one should keep in mind that it might still be necessary to get both zones delegated since old libraries are still asking for ip6.int. In the (hopefully) near future it will be sufficient to stick with `ip6.arpa.` only.

10.2 Adding IPv6 glue records for ccTLD nameservers

10.2.1 The Administrative Process

ICANN (http://www.icann.org) is the organization that currently ensures the IANA (http://www.iana.org) functions are carried out at a global level. One of IANA's functions is the DNS top-level delegations. This means any change in the country-code top level domain (ccTLD) nameservers has to be approved and activated by IANA. This is also true for other types of TLDs, namely the generic ones, that comprise the well known ".COM", ".NET", ".ORG" top-level domains.

The introduction of IPv6-capable nameservers on the DNS hierarchy at the ccTLD level has to be made through a request to IANA, from the registered contact for each domain. On http://www.iana.org/cctld/cctld-whois.htm, the entire list of countries and their associated contacts can be found. As an example, when FCCN, the sponsoring organization for Portugal, wanted to modify existent data about the nameservers supporting the .PT domain, the first step was an email from the administrative contact (Pedro Veiga) to ICANN (root-mgmt@icann.org).

10.2.2 The Technical Process

In technical terms, the changes in the DNS root zone for the specific case referred above were two:

- the name for a secondary server from the french ccTLD (nic.fr), that is at the same time supporting the .PT domain;
- the AAAA glue record for ns2.dns.pt (located at Oporto) was added.

In general terms, a ccTLD manager should consider defining how many of the servers supporting its own domain should carry AAAA records. Conservative approaches have been put in place by some countries in the end of 2004 by inserting only one or two IPv6 glue records. The name used in the SOA record of a ccTLD is normally not associated with any AAAA. This practice may evolve over time, and most servers should have their AAAA record visible on the DNS tree.

There is also a recommendation not to use long server names, because of a legacy 1024-bytes limit issue in DNS responses, which made some ccTLDs rename their servers to "A", "B", and "C" (which is the same philosophy being used for years in the root zone).

After this step, the process itself with IANA is rather simple and fast, if they can properly check all the nameservers using IPv4 and IPv6 addresses. At this time, this may be somewhat a problem, because IPv6 global routing may bring some difficulties. A ccTLD should really ensure global IPv6 reachability to their nameservers, and IANA's IPv6 transit is the second key factor for the tests to workout ok. In the portuguese case, the fact that IANA was still using `3FFE` addresses caused some problems. Some simple route debugging and IPv6 BGP tweaking solved it.

The state of IPv6 glue-records on TLD supporting nameservers can be checked using a simple unix tool:

```
dig @f.root-servers.net . axfr
```

This command has 4 components:

- "`dig`" is the tool itself;

- "`@f.root-servers.net`" is the server that permits everybody to get the complete root zone content (most of the other root servers deny this!);
- "`.`" is the root zone specification;
- "`axfr`" is the transfer zone type of request.

At the time this text is being written, 33 AAAA records are present in the DNS root zone, supporting 45 top-level domains.

10.2.3 Open Gaps

Introducing IPv6 availability on ccTLDs' nameservers is an important step to start full usage of IPv6 in the DNS tree. However, at the end of 2004 there are still some known gaps.

The 13 DNS root servers which contain the references to the ccTLDs' servers are still missing their IPv6 addresses on the reference files distributed with DNS servers' software (namely `named.ca` on BIND). Moreover, not all the 13 servers already have IPv6 enabled and globally reachable via IPv6.

Through http://www.root-servers.org, at the end of 2004, we can only see 6 of them with public IPv6 addresses defined.

The DNS structure will work fine in IPv6 only having a subset of these critical 13 root servers using it (not to forget that some of them are heavily replicated around the world). However, it is desirable to see all the servers dealing with IPv6, at least from the performance perspective.

11 References

[PIM-SM] "Protocol Independent Multicast – Sparse Mode (PIM-SM): Protocol Specification Revised", draft-ietf-pim-sm-v2-new-11.txt; Bill Fenner, Mark Handley, Hugh Holbrook, Isidor Kouvelas

[D3.5.1] "6NET Deliverable 3.5.1: Implementation of Security Plan"; October 2003

[D3.5.1v2] "6NET Deliverable 3.5.1v2: Implementation of Security Plan Version 2"; 2004

[RFC2842] "Capabilities Advertisement with BGP-4", RFC2842, R. Chandra, J. Scudder; May 2000

[RFC2874] "DNS Extensions to Support IPv6 Address Aggregation and Renumbering", RFC2874, M. Crawford, C. Huitema; July 2000

[RFC3152] "Capabilities Advertisement with BGP-4", RFC3152, R. Bush; August 2001

[RFC3363] "Representing Internet Protocol version 6 (IPv6) Addresses in the Domain Name System (DNS)", RFC3363, R. Bush, A. Durand, B. Fink, O. Gudmundsson, T. Hain (editors); August 2002

[RFC3364] "Tradeoffs in Domain Name System (DNS) Support for Internet Protocol version 6 (IPv6)", RFC3364, R. Austein; August 2002

[TNC2004-IPv6-FW] "Security for IPv6: From a Firewall's Perspective", Presentation at the annual Terena Networking Conference 2004 by Janos Mohacsi. Slides and Paper available at http://www.terena.nl/conferences/2004/programme/presentations/