XORP User Manual

Version 1.1

XORP Project
http://www.xorp.org/
feedback@xorp.org

April 13, 2005

Preface

This User Manual describes the process of configuring and operating a router running XORP. At the time of writing, XORP is a work-in-progress, and is evolving relatively quickly. We hope this user manual accurately reflects the functionality available in XORP, but it is likely to date quite quickly. An up-to-date copy of this manual will always be available from *http://www.xorp.org/*.

Contributing to this Manual

XORP is an open-source project, and this manual is an open-source manual. Like the XORP software, it is covered by the XORP license, which permits you to modify it and use it for any purpose whatsoever, so long as the copyright is preserved. Please help us improve this manual by sending contributions, suggestions, and criticism to *feedback@xorp.org*.

The XORP License

Contents

Glossary

AS: see Autonomous System.

Autonomous System: a routing domain that is under one administrative authority, and which implements its own routing policies. Key concept in BGP.

BGP: Border Gateway Protocol. See Chapter 8.

Bootstrap Router: A PIM-SM router that chooses the RPs for a domain from amongst a set of candidate RPs.

BSR: See Bootstrap Router.

Candidate RP: A PIM-SM router that is configured to be a candidate to be an RP. The Bootstrap Router will then choose the RPs from the set of candidates.

Dynamic Route: A route learned from another router via a routing protocol such as RIP or BGP.

EGP: see Exterior Gateway Protocol.

Exterior Gateway Protocol: a routing protocol used to route between Autonomous Systems. The main example is BGP.

IGMP: Internet Group Management Protocol. See Chapter 10.

IGP: see Interior Gateway Protocol.

Interior Gateway Protocol: a routing protocol used to route within an Autonomous System. Examples include RIP, OSPF and IS-IS.

Live CD: A CD-ROM that is bootable. In the context of XORP, the Live CD can be used to produce a low-cost router without needing to install any software.

MLD: Multicast Listener Discovery protocols. See Chapter 10.

MRIB: See Multicast RIB.

Multicast RIB: the part of the RIB that holds multicast routes. These are not directly used for forwarding, but instead are used by multicast routing protocols such as PIM-SM to perform RPF checks when building the multicast distibution tree.

OSPF: See Open Shortest Path First.

Open Shortest Path First: an IGP routing protocol based on a link-state algorithm. Used to route within medium to large networks.

PIM-SM: Protocol Independent Multicast, Sparse Mode. See Chapter 11.

Rendezvous Point: A router used in PIM-SM as part of the rendezvous process by which new senders are grafted on to the multicast tree.

Reverse Path Forwarding: many multicast routing protocols such as PIM-SM build a multicast distribution tree based on the best route back from each receiver to the source, hence multicast packets will be forwarded along the reverse of the path to the source.

RIB: See Routing Information Base.

RIP: Routing Information Protocol. See Chapter 7.

Routing Information Base: the collection of routes learned from all the dynamic routing protocols running on the router. Subdivided into a Unicast RIB for unicast routes and a Multicast RIB.

RP: See Rendezvous Point.

RPF: See Reverse Path Forwarding.

Static Route: A route that has been manually configured on the router.

xorpsh: XORP command shell. See Chapter 1.

xorp_rtrmgr: XORP router manager process. See Chapter 1.

Chapter 1

Command Structure

1.1 Introduction

To interact with a XORP router using the command line interface (CLI), the user runs the XORP command shell "xorpsh". This allows configuration of the router and monitoring of the router state.

In this chapter we describe how to interact with xorpsh. In later chapters we describe the details of how to configure BGP, PIM, SNMP and other processes.

The user interface style is loosely modelled on that of a Juniper router. This manual and the xorpsh itself are works in progress, and so may change significantly in the future.

1.2 Running xorpsh

The xorpsh command provides an interactive command shell to a XORP user, similar in many ways to the role played by a Unix shell. In a production router or on the XORP LiveCD, xorpsh might be set up as an user's login shell - they would login to the router via ssh and be directly in the xorpsh environment. However, for research and development purposes, it makes more sense to login normally to the machine running the XORP processes, and to run xorpsh directly from the Unix command line.

xorpsh should normally be run as a regular user; it is neither necessary or desirable to run it as root. If an user is to be permitted to make changes to the running router configuration, that user needs to be in the Unix group `xorp`. The choice of GID for group `xorp` is not important.

xorpsh needs to be able to communicate with the XORP router management process xorp_rtrmgr using the local file system. If the xorp_rtrmgr cannot write files in /tmp that xorpsh can read, then xorpsh will not be able to authenticate the user to the xorp_rtrmgr.

Multiple users can run xorpsh simultaneously. There is some degree of configuration locking to prevent simultaneous changes to the router configuration, but currently this is fairly primitive.

1.3 Basic Commands

On starting xorpsh, you will be presented with a command line prompt:

```
Xorp>
```

You can exit xorpsh at any time by trying Control-d.

Typing “?” at the prompt will list the commands currently available to you:

```
Xorp> ?
Possible completions:
  configure   Switch to configuration mode
  help        Provide help with commands
  quit        Quit this command session
  show        Display information about the system
```

If you type the first letter or letters of a command, and hit `<Tab>`, then command completion will occur.

At any time you can type “?” again to see further command completions. For example:

```
Xorp> config?
Possible completions:
  configure   Switch to configuration mode
Xorp> config
```

If the cursor is after the command, typing “?” will list the possible parameters for the command:

```
Xorp> configure ?
Possible completions:
  <[Enter]>   Execute this command
  exclusive   Switch to configuration mode, locking out other users
Xorp> configure
```

1.3.1 Command History and Command Line Editing

xorpsh supports emacs-style command history and editing of the text on the command line. The most important commands are:

- The **up-arrow** or **control-p** moves to the previous command in the history.
- The **down-arrow** or **control-n** moves to the next command in the history.
- The **left-arrow** or **control-b** moves back along the command line.
- The **right-arrow** or **control-f** move forward along the command line.
- **control-a** moves to the beginning of the command line.
- **control-e** moves to the end of the command line.
- **control-d** deletes the character directly under the cursor.
- **control-t** toggles (swaps) the character under the cursor with the character immediately preceding it.
- **control-space** marks the current cursor position.
- **control-w** deletes the text between the mark and the current cursor position, copying the deleted text to the cut buffer.
- **control-k** kills (deletes) from the cursor to the end of the command line, copying the deleted text to the cut buffer.
- **control-y** yanks (pastes) the text from the cut buffer, inserting it at the current cursor location.

1.4 Command Modes

xorpsh has two command modes:

Operational Mode, which allows interaction with the router to monitor its operation and status.

Configuration Mode, which allows the user to view the configuration of the router, to change that configuration, and to load and save configurations to file.

Generally speaking, operational mode is considered to give non-privileged access; there should be nothing an user can type that would seriously impact the operation of the router. In contrast, configuration mode allows all aspects of router operation to be modified.

In the long run, xorpsh and the xorp_rtrmgr will probably come to support fine-grained access control, so that some users can be given permission to change only subsets of the router configuration. At the present time though, there is no fine-grained access control.

An user can only enter configuration mode if that user is in the `xorp` Unix group.

1.5 Operational Mode

```
Xorp> ?
Possible completions:
  configure   Switch to configuration mode
  help        Provide help with commands
  quit        Quit this command session
  show        Display information about the system
```

The main commands in operational mode are:

configure: switches from operational mode to configuration mode.

help: provides online help.

quit: quit from xorpsh.

show: displays many aspects of the running state of the router.

1.5.1 Show Command

```
Xorp> show ?
Possible completions:
  <[Enter]>   Execute this command
  bgp         Display information about BGP
  host        Display information about the host
  igmp        Display information about IGMP
  interfaces  Show network interface information
  mfea        Display information about IPv4 MFEA
  mfea6       Display information about IPv6 MFEA
  mld         Display information about MLD
  pim         Display information about IPv4 PIM
  pim6        Display information about IPv6 PIM
  rip         Display information about RIP
  route       Show route table
Xorp> show
```

The *show* command is used to display many aspects of the running state of the router. We don't describe the sub-commands here, because they depend on the running state of the router. For example, only a router that is running BGP should provide `show bgp` commands.

As an example, we show the peers of a BGP router:

```
Xorp> show bgp peers detail
OK
Peer 1: local 192.150.187.108/179 remote 192.150.187.109/179
  Peer ID: 192.150.187.109
  Peer State: ESTABLISHED
  Admin State: START
  Negotiated BGP Version: 4
  Peer AS Number: 65000
  Updates Received: 5157, Updates Sent: 0
  Messages Received: 5159, Messages Sent: 1
  Time since last received update: 4 seconds
  Number of transitions to ESTABLISHED: 1
  Time since last entering ESTABLISHED state: 47 seconds
  Retry Interval: 120 seconds
  Hold Time: 90 seconds, Keep Alive Time: 30 seconds
  Configured Hold Time: 90 seconds, Configured Keep Alive Time: 30 seconds
  Minimum AS Origination Interval: 0 seconds
  Minimum Route Advertisement Interval: 0 seconds
```

1.6 Configuration Mode

```
Xorp> configure
Entering configuration mode.
There are no other users in configuration mode.
[edit]
XORP>
```

When in configuration mode, the command prompt changes to be all capitals. The command prompt is also usually preceded by a line indicating which part of the configuration tree is currently being edited.

```
[edit]
XORP> ?
Possible completions:
  create          Create a sub-element
  delete          Delete a configuration element
  edit            Edit a sub-element
  exit            Exit from this configuration level
  help            Provide help with commands
  load            Load configuration from a file
  quit            Quit from this level
  run             Run an operational-mode command
  save            Save configuration to a file
  set             Set the value of a parameter
  show            Show the value of a parameter
  top             Exit to top level of configuration
  up              Exit one level of configuration
XORP>
```

The router configuration has a tree form similar to the directory structure on a Unix filesystem. The current configuration or parts of the configuration can be shown with the *show* command:

```
[edit]
XORP> show interfaces
  interface rl0 {
    description: "control interface"
    vif rl0 {
      address 192.150.187.108 {
        prefix-length: 25
        broadcast: 192.150.187.255
        disable: false
      }
      disable: false
    }
    disable: false
  }
  targetname: "fea"
```

1.6.1 Moving around the Configuration Tree

You can change the current location in the configuration tree using the *edit*, *exit*, *quit*, *top* and *up* commands.

- **edit** <***element name***>: Edit a sub-element
- **exit**: Exit from this configuration level, or if at top level, exit configuration mode.
- **quit**: Quit from this level
- **top**: Exit to top level of configuration
- **up**: Exit one level of configuration

For example:

```
[edit]
XORP> edit interfaces interface rl0 vif rl0
[edit interfaces interface rl0 vif rl0]
XORP> show
  address 192.150.187.108 {
    prefix-length: 25
    broadcast: 192.150.187.255
    disable: false
  }
  disable: false

[edit interfaces interface rl0 vif rl0]
XORP> up
[edit interfaces interface rl0]
XORP> top
[edit]
XORP>
```

1.6.2 Loading and Saving Configurations

On startup, the xorp_rtrmgr will read a configuration file. It will then start up and configure the various router components as specified in the configuration file.

The configuration file can be created externally, using a normal text editor, or it can be saved from the running router configuration. A configuration file can also be loaded into a running router, causing the previous running configuration to be discarded. The commands for this are:

- **save** <***filename***>: save the current configuration in the specified file.
- **load** <***filename***>: load the specified file, discarding the currently running configuration.

1.6.3 Setting Configuration Values

- **set** <***path to config***> <***value***>: set the value of the specified configuration node.

The *set* command is used to set or change the value of a configuration option. The change does not actually take effect immediately - the *commit* command must be used to apply this and any other uncommitted changes.

In the example below, the prefix length (netmask) of address 192.150.187.108 on vif rl0 is changed, but not yet committed. The “>” indicates parts of the configuration that have changed but not yet been committed.

```
[edit interfaces interface rl0]
XORP> show
  description: "control interface"
  vif rl0 {
    address 192.150.187.108 {
      prefix-length: 25
      broadcast: 192.150.187.255
      disable: false
    }
    disable: false
  }
  disable: false

[edit interfaces interface rl0]
XORP> set vif rl0 address 192.150.187.108 prefix-length 24
OK

[edit interfaces interface rl0]
XORP> show
  description: "control interface"
  vif rl0 {
    address 192.150.187.108 {
>     prefix-length: 24
      broadcast: 192.150.187.255
      disable: false
    }
    disable: false
  }
```

1.6.4 Adding New Configuration

- **create** <***path to new config node***> : create new configuration node.
- **create** <***path to new config node***> { : create new configuration node and start editing it.

New configuration can be added by the *create* command. If we type *create* followed by the path to a new configuration node, the node will be created. All parameters within that node will be assigned their default values (if exist). After that the node can be edited with the *edit* command. If we type { after the path to the new configuration node, the node will be created, the default values will be assigned, and we can directly start editing that node. The user interface for this is currently rather primitive and doesn't permit the more free-form configuration allowed in configuration files.

For example, to configure a second vif on interface rl0:

```
[edit interfaces interface rl0]
XORP> show
  description: "control interface"
  vif rl0 {
    address 192.150.187.108 {
      prefix-length: 24
      broadcast: 192.150.187.255
      disable: false
    }
    disable: false
  }
  disable: false

[edit interfaces interface rl0]
XORP> create vif rl0b {
    > address 10.0.0.1 {
    >   prefix-length 16
    >   broadcast 10.0.255.255
    >   disable false
    >   }
    > disable false
    > }
[edit interfaces interface rl0]
XORP> show
  description: "control interface"
  vif rl0 {
    address 192.150.187.108 {
      prefix-length: 24
      broadcast: 192.150.187.255
      disable: false
    }
    disable: false
  }
> vif rl0b {
>   address 10.0.0.1 {
>     prefix-length: 16
>     broadcast: 10.0.255.255
>     disable: false
>   }
>   disable: false
> }
  disable: false
```

1.6.5 Deleting Parts of the Configuration

The *delete* command can be used to delete subtrees from the configuration. The deletion will be visible in the results of the *show* command, but will not actually take place until the changes are committed.

```
XORP> show interfaces interface rl0
  description: "control interface"
  vif rl0 {
    address 192.150.187.108 {
      prefix-length: 24
      broadcast: 192.150.187.255
      disable: false
    }
    disable: false
  }
  vif rl0b {
    address 10.0.0.1 {
      prefix-length: 16
      broadcast: 10.0.255.255
      disable: false
    }
    disable: false
  }
  disable: false

[edit]
XORP> delete interfaces interface rl0 vif rl0b
Deleting:
  address 10.0.0.1 {
    prefix-length: 16
    broadcast: 10.0.255.255
    disable: false
  }

OK
[edit]
XORP> show interfaces interface rl0
  description: "control interface"
  vif rl0 {
    address 192.150.187.108 {
      prefix-length: 24
      broadcast: 192.150.187.255
      disable: false
    }
    disable: false
  }
  disable: false
```

1.6.6 Committing Changes

```
[edit interfaces interface rl0]
XORP> commit
OK
```

The *commit* command commits all the current configuration changes. This can take a number of seconds before the response is given.

If xorpsh was built with debugging enabled, the response can be considerably more verbose than shown above!

If two or more users are logged in using configuration mode, and one of them changes the configuration, the others will receive a warning:

```
[edit]
XORP>
The configuration had been changed by user mjh
XORP>
```

1.6.7 Discarding Changes

The user can discard a batch of changes by editing them back to their original configuration, or by using the *exit* command to leave configuration mode:

```
[edit]
XORP> exit
ERROR: There are uncommitted changes
Use "commit" to commit the changes, or "exit discard" to discard them
XORP> exit discard
Xorp>
```

Chapter 2

Configuration Overview

2.1 Introduction

A XORP router must be configured to perform the desired operations. The configuration information can be provided in one of the two ways:

- Use a configuration file when the rtrmgr is started. By default, the rtrmgr will load the configuration from file "config.boot" in the XORP installation directory. This file can be specified by the "-b <filename>" command line option:

  ```
  xorp_rtrmgr -b my_config.boot
  ```

  See "rtrmgr/config.boot.sample" for an example of a configuration file (note that this file MUST be modified before using it).

- Use the xorpsh command line interface after the rtrmgr is started. It should be noted that command line completion in the xorpsh does greatly simplify configuration.

A mixture of both methods is permissible. For example, a configuration file can also be loaded from within the xorpsh.

At very minimum, a router's interfaces must be configured (see Section 2.2). Typically, the FEA needs to be configured (*e.g.,* to enable unicast forwarding); the FEA configuration is described in Section 2.3. All protocol configuration is described in Section 2.4.

2.2 Network Interfaces

A XORP router will only use interfaces that it has been explicitly configured to use. Even for protocols such as BGP that are agnostic to interfaces, if the next-hop router for a routing entry is not through a configured interface the route will not be installed. For protocols that are explicitly aware of interfaces only configured interfaces will be used.

Every physical network device in the system is considered to be an "interface". Every interface can contain a number of virtual interfaces ("vif"s). In the majority of cases the interface name and vif name will be identical and will map to the name given to the interface by the operating system. A virtual interface is configured with the address or addresses that should be used. At each level in the configuration hierarchy (`interface`, `vif` and `address`) it is necessary to enable this part of the configuration.

```
interfaces {
  interface dc0 {
    description: "data interface"
    disable: false
    /* default-system-config */
    vif dc0 {
      disable: false
      address 10.10.10.10 {
        prefix-length: 24
        broadcast: 10.10.10.255
        disable: false
      }
      /*
      address 10:10:10:10:10:10:10:10 {
        prefix-length: 64
        disable: false
      }
      */
    }
  }
}
```

We recommend that you select the interfaces that you want to use on your system and configure them as above. If you are configuring an interface that is currently being used by the the system make sure that there is no mismatch in the `address`, `prefix-length` and `broadcast` arguments. If the `default-system-config` statement is used, it instructs the FEA that the interface should be configured by using the existing interface information from the underlying system. In that case, the `vif` and `address` sections must not be configured.

2.3 Forwarding Engine Abstraction

It is a requirement to explicitly enable forwarding for each protocol family.

```
fea {
  unicast-forwarding4 {
    disable: false
  }
/*
  unicast-forwarding6 {
    disable: false
  }
*/
}
```

If IPv4 forwarding is required you will require the configuration above. If the system supports IPv6 and IPv6 forwarding is required, then the `unicast-forwarding6` statement must be used to enable it [1].

[1] Note that prior to XORP Release-1.1, the `enable-unicast-forwarding4` and `enable-unicast-forwarding6` flags were used instead to enable or disable the IPv4 and IPv6 forwarding.

2.4 Protocols

A unicast router typically will be configured with one or more of the following protocols: StaticRoutes (Section 2.4.1), RIP (Section 2.4.2) or BGP (Section 2.4.3).

A multicast router must have the MFEA configured (Section 2.4.4). Typically, a multicast router should have IGMP/MLD configured (Section 2.4.5). Currently, PIM-SM is the only multicast routing protocol implemented (Section 2.4.6). If some multicast-specific static routes need to be installed in the MRIB (for computing the reverse-path forwarding information), those can be specified in the StaticRoutes configuration (Section 2.4.1). If there are no unicast routing protocols configured, the FIB2MRIB module may need to be configured as well (Section 2.4.7).

2.4.1 Static Routes

This is the simplest routing protocol in XORP. It allows the installation of unicast or multicast static routes (either IPv4 or IPv6). Note that in case of multicast the routes are installed only in the user-level Multicast Routing Information Base and are used for multicast-specific reverse-path forwarding information by multicast routing protocols such as PIM-SM.

```
protocols {
  static {
    route4 10.20.0.0/16 {
      nexthop: 10.10.10.20
      metric: 1
    }
    mrib-route4 10.20.0.0/16 {
      nexthop: 10.10.10.30
      metric: 1
    }
    /*
    route6 20:20:20:20::/64 {
      nexthop: 10:10:10:10:10:10:10:20
      metric: 1
    }
    mrib-route6 20:20:20:20::/64 {
      nexthop: 10:10:10:10:10:10:10:30
      metric: 1
    }
    */
  }
}
```

2.4.2 Routing Information Protocol

In order to run RIP it is sufficient to specify the set of interfaces, vifs and addresses (`interface`, `vif` and `address`) on which RIP is enabled [2].

If you wish to announce routes then it is necessary to `export` [3] the routes that are to be announced. For example, `connected`, `static` and `rip`.

Note that this mechanism for originating routes is temporary. In the next release it will be replaced by a more generic policy framework. Unfortunately, the same keyword `export` will be used but with different syntax.

```
protocols {
  rip {
  /* Redistribute routes for connected interfaces */
    /*
    export connected {
      metric: 0
      tag: 0
    }
    */
  /* Redistribute static routes */
    /*
    export static {
      metric: 1
      tag: 0
    }
    */
  /* Run on specified network interface addresses */
  /*
    interface dc0 {
      vif dc0 {
        address 10.10.10.10 {
          disable: false
        }
      }
    }
  */
  }
}
```

[2]Note that prior to XORP Release-1.1, the `enable` flag was used instead of `disable` to enable or disable each part of the configuration.

[3]Note that in release candidate 1.0-RC, the `import` keyword is used in place of `export`.

2.4.3 Border Gateway Protocol

In order to run BGP the `bgp-id` (BGP Identifier) and `local-as` (Autonomous System number) must be specified.

The `peer` statement specifies a peering. The argument to the peer statement is the IP address of the peer. The `local-ip` is the IP address that TCP should use. The `as` is the Autonomous System Number of the peer.

```
protocols {
  bgp {
    bgp-id: 10.10.10.10
    local-as: 65002

    peer 10.30.30.30 {
      local-ip: 10.10.10.10
      as: 65000
      next-hop: 10.10.10.20
      /*
      local-port: 179
      peer-port: 179
      */
      /* holdtime: 120 */
      /* disable: false */

      /* IPv4 unicast is enabled by default */
      /* ipv4-unicast: true */

      /* Optionally enable other AFI/SAFI combinations */
      /* ipv4-multicast: true */
      /* ipv6-unicast: true */
      /* ipv6-multicast: true */
    }

    /* Originate IPv4 Routes */
    /*
    network4 10.10.10.0/24 {
      next-hop: 10.10.10.10
      unicast: true
      multicast: true
    }
    */

    /* Originate IPv6 Routes */
    /*
    network6 10:10:10:10::/64 {
      next-hop: 10:10:10:10:10:10:10:10
      unicast: true
      multicast: true
    }
    */
  }
}
```

Currently BGP is not able to `import` routes from other routing protocols such as `static`. It is however possible to originate routes using `network4` and `network6` statements such as in the above example.

2.4.4 Multicast Forwarding Engine Abstraction

The MFEA must be configured if the XORP router is to be used for multicast routing. The MFEA for IPv4 and IPv6 are configured separately.

In the configuration we must explicitly configure the entity itself, and each `vif`. The `traceoptions` section is used to explicitly enable log information that can be used for debugging purpose [4].

```
plumbing {
  mfea4 {
    disable: false
    interface dc0 {
      vif dc0 {
        disable: false
      }
    }
    interface register_vif {
      vif register_vif {
        /* Note: this vif should be always enabled */
        disable: true
      }
    }
    traceoptions {
      flag all {
        disable: true
      }
    }
  }
}

plumbing {
  mfea6 {
    disable: true
    interface dc0 {
      vif dc0 {
        disable: true
      }
    }
    interface register_vif {
      vif register_vif {
        /* Note: this vif should be always enabled */
        disable: true
      }
    }
    traceoptions {
      flag all {
        disable: true
      }
    }
  }
}
```

Note that the interface/vif named `register_vif` is special. If PIM-SM is configured, then `register_vif` must be enabled in the MFEA.

[4]Note that prior to XORP Release-1.1, the `enable` flag was used instead of `disable` to enable or disable each part of the configuration.

2.4.5 Internet Group Management Protocol/Multicast Listener Discovery

IGMP/MLD should be configured if the XORP router is to be used for multicast routing and if we want to track multicast group membership for directly connected subnets. Typically this is the case for a multicast router, therefore it should be enabled. IGMP and MLD are configured separately: IGMP is used for tracking IPv4 multicast members; MLD is used for tracking IPv6 multicast members.

In the configuration we must explicitly configure each entity and each `vif`. The `traceoptions` section is used to explicitly enable log information that can be used for debugging purpose [5].

```
protocols {
  igmp {
    disable: false
    interface dc0 {
      vif dc0 {
        disable: false
      }
    }
    traceoptions {
      flag all {
        disable: false
      }
    }
  }
}

protocols {
  mld {
    disable: false
    interface dc0 {
      vif dc0 {
        disable: false
      }
    }
    traceoptions {
      flag all {
        disable: false
      }
    }
  }
}
```

[5]Note that prior to XORP Release-1.1, the `enable` flag was used instead of `disable` to enable or disable each part of the configuration.

2.4.6 Protocol Independent Multicast - Sparse Mode

PIM-SM should be configured if the XORP router is to be used for multicast routing in PIM-SM domain. PIM-SM for IPv4 and IPv6 are configured separately. At minimum, the entity itself and the virtual interfaces should be enabled, and the mechanism for obtaining the Candidate-RP set (either the Bootstrap mechanism, or a static-RP set) [6].

```
protocols {
  pimsm4 {
    disable: false
    interface dc0 {
      vif dc0 {
        disable: false
        /* dr-priority: 1 */
        /* alternative-subnet 10.40.0.0/16 */
      }
    }
    interface register_vif {
      vif register_vif {
        /* Note: this vif should be always enabled */
        disable: false
      }
    }

    static-rps {
      rp 10.60.0.1 {
        group-prefix 224.0.0.0/4 {
          /* rp-priority: 192 */
          /* hash-mask-len: 30 */
        }
      }
    }

    bootstrap {
      disable: false
      cand-bsr {
        scope-zone 224.0.0.0/4 {
          /* is-scope-zone: false */
          cand-bsr-by-vif-name: "dc0"
          /* bsr-priority: 1 */
          /* hash-mask-len: 30 */
        }
      }

      cand-rp {
        group-prefix 224.0.0.0/4 {
          /* is-scope-zone: false */
          cand-rp-by-vif-name: "dc0"
          /* rp-priority: 192 */
          /* rp-holdtime: 150 */
        }
      }
    }
```

continued overleaf....

[6]Note that prior to XORP Release-1.1, the `enable` flag was used instead of `disable` to enable or disable each part of the configuration.

```
    switch-to-spt-threshold {
      /* approx. 1K bytes/s (10Kbps) threshold */
      disable: false
      interval-sec: 100
      bytes: 102400
    }

    traceoptions {
      flag all {
        disable: false
      }
    }
  }
}

protocols {
  pimsm6 {
    disable: false
    interface dc0 {
      vif dc0 {
        disable: false
        /* dr-priority: 1 */
        /* alternative-subnet 40:40:40:40::/64 */
      }
    }
    interface register_vif {
      vif register_vif {
        /* Note: this vif should be always enabled */
        disable: false
      }
    }

    static-rps {
      rp 50:50:50:50:50:50:50:50 {
        group-prefix ff00::/8 {
          /* rp-priority: 192 */
          /* hash-mask-len: 126 */
        }
      }
    }

    bootstrap {
      disable: false
      cand-bsr {
        scope-zone ff00::/8 {
          /* is-scope-zone: false */
          cand-bsr-by-vif-name: "dc0"
          /* bsr-priority: 1 */
          /* hash-mask-len: 30 */
        }
      }

      cand-rp {
        group-prefix ff00::/8 {
          /* is-scope-zone: false */
          cand-rp-by-vif-name: "dc0"
          /* rp-priority: 192 */
          /* rp-holdtime: 150 */
        }
      }
    }
```

continued overleaf....

```
    switch-to-spt-threshold {
      /* approx. 1K bytes/s (10Kbps) threshold */
      disable: false
      interval-sec: 100
      bytes: 102400
    }

    traceoptions {
      flag all {
        disable: false
      }
    }
  }
}
```

A number of parameters have default values, therefore they don't have to be configured (those parameters are commented-out in the above sample configuration).

Note that the interface/vif named `register_vif` is special. If PIM-SM is configured, then `register_vif` must be enabled.

The `dr-priority` parameter is used to configure the Designated Router priority per virtual interface (note that in case of `register_vif` it is not used).

The `alternative-subnet` statement is used to associate additional subnets with a network interface. For example, if you want to make incoming traffic with a non-local source address appear as it is coming from a local subnet, then `alternative-subnet` can be used. Typically, this is needed as a work-around solution when we use uni-directional interfaces for receiving traffic (e.g., satellite links). Note: use `alternative-subnet` with extreme care, only if you know what you are really doing!

If PIM-SM uses static RPs, those can be configured within the `static-rps` section. For each RP, an `rp` section is needed, and each section should contain the multicast prefix address the static RP is configured with. The RP priority can be modified with the `rp-priority` parameter.

If PIM-SM uses the Bootstrap mechanism to obtain the Candidate-RP set, this can be configured in the `bootstrap` section. If the XORP router is to be used as a Candidate-BSR, this should be specified in the `cand-bsr` section. For a router to be a Candidate-BSR it must advertise for each zone (scoped or non-scoped) the associated multicast prefix address. The `cand-bsr` section should contain `scope-zone` statements for each multicast prefix address. The vif name with the address that is to be used as the Candidate-BSR is specified by the `cand-bsr-by-vif-name` statement. The Candidate-BSR priority can be modified with the `bsr-priority` parameter.

If the XORP router is to be a Candidate-RP, this should be specified in the `cand-rp` section. For a router to be a Candidate-RP it must advertise for each zone (scoped or non-scoped) the associated multicast prefix address. The `cand-rp` section should contain `group-prefix` statements for each multicast prefix address. The vif name with the address that is to be used as the Candidate-RP is specified by the `cand-rp-by-vif-name` statement. The Candidate-RP priority can be modified with the `rp-priority` parameter; the Candidate-RP holdtime can be modified with the `rp-holdtime` parameter.

The `is-scope-zone` parameter is used to specify whether a Candidate-BSR `scope-zone` or a Candidate-RP `group-prefix` is scoped. Currently, scoped zones are not well tested, hence it is recommended `scope-zone` is always set to `false`. Note that typically the `hash-mask-len` should not be modified; if you don't know what `hash-mask-len` is used for, don't modify it!

The `switch-to-spt-threshold` section can be used to specify the multicast data bandwidth threshold used by the last-hop PIM-SM routers and the RPs to initiate shortest-path switch toward the multicast source. Parameter `interval-sec` is used to specify the periodic measurement interval; parameter `bytes` is used to specify the threshold in number of bytes within the measurement interval. It is recommended that the measurement interval is not too small, and should be on the order of tens of seconds.

The `traceoptions` section is used to explicitly enable log information that can be used for debugging purpose.

2.4.7 FIB2MRIB

The FIB2MRIB module is used to obtain the Forwarding Information Base information from the underlying system (via the FEA), and to propagate it to the MRIB, so it can be used by multicast routing protocols such as PIM-SM. Typically, it is needed only if the unicast routing protocols (if any) on that router do not inject routes into the MRIB [7].

```
protocols {
  fib2mrib {
    disable: false
  }
}
```

[7]Note that prior to XORP Release-1.1, the `enable` flag was used instead of `disable` to enable or disable each part of the configuration.

Chapter 3

Network Interfaces

3.1 Network Interfaces Terminology and Concepts

A router receives packets via its network interfaces from its neighboring routers. Some of those packets will be destined for the router itself, but most of then will normally be forwarded on via another network interface to another router or to locally connected hosts.

There are many different types of network interface, such as Ethernet, ATM, DS3, or ISDN. Sometimes the underlying physical interface will need explicit configuration before it can establish a link, and sometimes the link requires no configuration. In addition, some network interfaces behave from a routing point of view as if they were really multiple interfaces, in that the router may have to forward packets between different "channels" on the same interface.

We choose to distinguish in a XORP router between physical interfaces (which we call *interfaces*, and logical interfaces, which we call virtual interfaces or *vifs*. An example of a *interface* might be an Ethernet card. An example of a *vif* might be one of many VLANs configured on that Ethernet[1].

Conceptually, XORP routes packets between vifs. Thus it is also vifs that are given IP addresses. Each interface may contain many vifs. Conversely every vif is always part of an interface, although some interfaces such as the loopback interface do not have a physical realisation.

The XORP naming convention for vifs is that they are named as they would be in the underlying forwarding path. For example, if the forwarding path is implemented in the FreeBSD kernel, then `fxp0` might denote a 100-base-T Ethernet vif (with no VLAN). On a router using Linux as the forwarding path, the same vif might be called `eth0`.

If a physical interface cannot support multiple vifs, or if there's a default vif on a physical interface, then the interface name and the vif name will normally be the same. Again, this is determined by the underlying forwarding path. A common example would be Ethernet without VLANs, where the interface and vif might both be named `fxp0` on FreeBSD or both called `eth0` on Linux.

[1]Currently (April 2005), VLANs are not yet supported in XORP.

3.2 Configuring Network Interfaces

A XORP router will only use interfaces that it has been explicitly configured to use. For protocols such as RIP that are explicitly aware of interfaces, only configured interfaces will be used. Even for protocols such as BGP that don't directly associate peerings with interfaces, if the next-hop router for a routing entry is not through a configured interface, the route will not be installed.

3.2.1 Configuration Syntax

The available configuration syntax for network interfaces and addresses is as follows:

```
interfaces {
  interface text {
    description: text
    mac: macaddr
    mtu: uint
    default-system-config
    disable: bool
    vif text {
      disable: bool
      address IPv4-addr {
        prefix-length: int(1..32)
        broadcast: IPv4-addr
        destination: IPv4-addr
        disable: bool
      }
      address IPv6-addr {
        prefix-length: int(1..128)
        destination: IPv6-addr
        disable: bool
      }
    }
  }
}
```

`interfaces`: this delimits all the interface configuration information within the XORP configuration file.

`interface`: this delimits the configuration for a particular interface. The parameter is the name of the interface, which must correspond to an interface known to the router forwarding path.

For each interface, the following configuration is possible:

`description`: this is a human-readable description for the interface. It is primarily used to help the router operator remember which interface serves which purpose. It is optional.

`mac`: This allows the MAC address for the interface to be set. MAC addresses on devices such as Ethernets are usually fixed, but in some cases it is possible to override the built-in default MAC address. The format should be in a form appropriate for the interface type. For an Ethernet interface, this would be six colon-separated 8-bit numbers in hexadecimal, such as `00:0a:59:9a:f2:ba`.

`mtu`: This allows the maximum transfer unit (MTU) to be set for the interface as a whole (applying to all VIFs). The value is an integer number of bytes, and should be less than or equal to the largest MTU supported by the physical device. When forwarding, IPv4 packets larger than the

MTU will be fragmented unless they the DF bit set, in which case they will be dropped and an ICMP Packet-too-big message will be returned to the sender.

`default-system-config`: Normally all the interfaces, vifs, and addresses on a XORP router will be configured through the XORP configuration file and command line interface. However, under certain circumstances it is useful to be able to run XORP as a routing daemon without changing the current configuration of interfaces and addresses. This primitive tells XORP to obtain its configuration for this interface by reading the existing configuration back from the forwarding engine rather than by configuring the forwarding engine. If `default-system-config` is used, then the `vif` and `address` sections must not be configured.

`disable`: this flag disables or enables the interface for routing and forwarding [2]. It takes the value `true` or `false`. Configuring an interface to be disabled has the same effect as removing its configuration, but without losing what the configuration used to be.

`vif`: this configures a vif on the corresponding interface. In some cases this may cause the vif to be created; an example might be an Ethernet VLAN. In other cases this merely denotes the start of the configuration for the vif. The parameter is the name of the vif, as understood by the router forwarding engine.

For each vif, the following configuration is possible:

`disable`: this flag disables or enables the vif for routing and forwarding [3]. It takes the value `true` or `false`. Configuring a vif to be disabled has the same effect as removing its configuration, but without losing what the configuration used to be.

`address`: this specifies a new IP address for this vif. A single vif might have multiple IP addresses, and might have both IPv4 address and IPv6 addresses. The parameter is either an IPv4 or IPv6 address.

For each address, the following configuration is possible:

`prefix-length`: this gives the prefix length of the subnet connected to this interface. For an IPv4 address, prefix-length must be between 4 and 32. For an IPv6 address, prefix-length must be between 8 and 128. This field is mandatory for each address.

`broadcast`: this gives the subnet broadcast address for the subnet corresponding to the vif address. It is only needed for IPv4 addresses (it is mandatory), and is needed for historical reasons. It takes the form of an IPv4 address.
Normally the broadcast address will have the local hosts part of the subnet address set to all ones. For example, with address 10.0.0.0 and prefix-length 20, the broadcast address will have the last 12 bits set to one, and hence will be 10.0.15.255.

`destination`: this specifies the destination IP address. It is only relevant for point-to-point interfaces, where the IP addresses at each end of the link need not share an IP subnet.

`disable`: this flag disables or enables this IP address on this vif [4]. It takes the value `true` or `false`. Configuring an IP address to be disabled has the same effect as removing its configuration, but without losing what the configuration used to be.

[2] Note that prior to XORP Release-1.1, the `enable` flag was used instead of `disable`.
[3] Note that prior to XORP Release-1.1, the `enable` flag was used instead of `disable`.
[4] Note that prior to XORP Release-1.1, the `enable` flag was used instead of `disable`.

3.2.2 Example Configurations

We recommend that you select the interfaces that you want to use on your system and configure them as below. Interfaces that you do not wish XORP to use for forwarding should be omitted from the configuration.

Configuring Interface Addresses

```
interfaces {
  interface dc0 {
    description: "ethernet interface"
    disable: false
    vif dc0 {
      disable: false
      address 10.10.10.10 {
        prefix-length: 24
        broadcast: 10.10.10.255
        disable: false
      }

      address 10:10:10:10:10:10:10:10 {
        prefix-length: 64
        disable: false
      }

    }
  }
}
```

In the example above, the router has only one interface configured. This interface is called `dc0`, and the vif is also called `dc0`. In this case, this is because this interface is an Ethernet inferface, and VLANs are not being used, so the vif is simply the default vif for this interface.

The vif has both an IPv4 and an IPv6 address configured. The IPv4 address is `10.10.10.10`, and connects to the subnet `10.10.10.0/24` as determined by the prefix-length. Consistent with this, the subnet broadcast address is `10.10.10.255`.

The IPv6 address has a prefix-length of 64 bits, and does not need (or allow) the broadcast address to be explicitly specified.

In this case, the internface is not a point-to-point interface, so no destination address is specified.

Using Pre-Configured Interface Addresses

If the `default-system-config` statement is used, as shown in the example belore, it instructs the FEA that the interface should be configured by using the existing interface information from the underlying system. In that case, the `vif` and `address` sections must not be configured.

```
interfaces {
  interface dc0 {
    description: "data interface"
    disable: false
    default-system-config
  }
}
```

3.3 Monitoring Network Interfaces

The state of a XORP router's interfaces can be displayed from operational mode using the `show interfaces` command. By itself, `show interfaces` will list information about all the interfaces in the router:

```
Xorp> show interfaces
dc0/dc0: Flags:<ENABLED,BROADCAST,MULTICAST> mtu 1500
        inet 172.16.0.1 subnet 172.16.0.0/24 broadcast 172.16.0.255
        physical index 1
        ether 00:80:c8:b9:61:09
dc1/dc1: Flags:<ENABLED,BROADCAST,MULTICAST> mtu 1500
        inet 172.16.1.1 subnet 172.16.1.0/24 broadcast 172.16.0.255
        physical index 2
        ether 00:80:c8:b9:61:0a
dc2/dc2: Flags:<ENABLED,BROADCAST,MULTICAST> mtu 1500
        inet 172.16.2.1 subnet 172.16.2.0/24 broadcast 172.16.0.255
        physical index 3
        ether 00:80:c8:b9:61:0b
dc3/dc3: Flags:<ENABLED,BROADCAST,MULTICAST> mtu 1500
        inet 172.16.3.1 subnet 172.16.3.0/24 broadcast 172.16.0.255
        physical index 4
        ether 00:80:c8:b9:61:0c
fxp0/fxp0: Flags:<ENABLED,BROADCAST,MULTICAST> mtu 1500
        inet 192.150.187.112 subnet 192.150.187.0/25 broadcast 192.150.187.255
        physical index 5
        ether 00:02:b3:10:b4:6c
```

In this case, the router has five Ethernet interfaces, each of which has a single vif. The naming format is *interface/vif*. For example `dc1/vlan2` would be vif vlan2 on interface dc1. In the above example, all the vif names are the same as the Ethernet interface names because no VLANs are being used.

To display information about a specific interface, use the `show interfaces` <*interface*> command:

```
Xorp> show interfaces dc1
dc1/dc1: Flags:<ENABLED,BROADCAST,MULTICAST> mtu 1500
        inet 172.16.1.1 subnet 172.16.1.0/24 broadcast 172.16.0.255
        physical index 2
        ether 00:80:c8:b9:61:0a
```

Chapter 4

Forwarding Engine

4.1 Terminology and Concepts

The forwarding engine is that part of a router that receives packets and forwards then from one interface to another. In the case of XORP, the forwarding engine may be the kernel forwarding path on Linux or FreeBSD, or it may reside in external forwarding hardware.

On any particular router, it might be desirable to enable or disable different parts of the forwarding functionality. For example, a router might only be intended to forward IPv6 packets but not IPv4 packets, or it might be intended to forward unicast packets but not multicast packets. Thus XORP provides the ability to enable and configure various forwarding functionality.

In XORP, the term "`fea`" refers to *Forwarding Engine Abstraction* and the term "`mfea`" refers to *Multicast Forwarding Engine Abstraction*. The term abstraction here refers to a high-level configuration interface that should be the same irrespective of whether the forwarding engine is provided in software in the operating system kernel or in external forwarding hardware.

4.2 Configuration of the Forwarding Engine

On a XORP router, forwarding functionality must be explicitly enabled or no packets will be forwarded. Forwarding can be separately enabled for unicast and multicast, and for IPv4 and IPv6. In addition, multicast interfaces/vifs need to be explicitly enabled individually, and certain special-purpose forwarding functionality can also be enabled for multicast.

4.2.1 Configuration Syntax

```
fea {
  targetname: txt
  unicast-forwarding4 {
    disable: bool
  }
  unicast-forwarding6 {
    disable: bool
  }
}
plumbing {
  mfea4 {
    disable: bool
    interface text {
      vif text {
        disable: bool
      }
    }
    interface register_vif {
      vif register_vif {
        disable: bool
      }
    }
    traceoptions {
      flag all {
        disable: bool
      }
    }
  }

  mfea6 {
    disable: bool
    interface text {
      vif text {
        disable: bool
      }
    }
    interface register_vif {
      vif register_vif {
        disable: bool
      }
    }
    traceoptions {
      flag {
        all {
          disable: bool
        }
      }
    }
  }
}
```

`fea`: this delimits the configuration for the unicast forwarding engine functionality.

The following unicast forwarding engine parameters can be configured:

- `targetname`: this is the name for this instance of the forwarding engine abstraction. It defaults to "`fea`", and it is strongly recommended that this default is *not* overridden under normal usage scenarios.

`unicast-forwarding4`: this directive is used to configure the IPv4 forwarding [1]. Possible parameters are:

- `disable`: this takes the value `true` or `false`, and disables or enables all IPv4 unicast forwarding on the router.

`unicast-forwarding6`: this directive is used to configure the IPv6 forwarding [2]. Possible parameters are:

- `disable`: this takes the value `true` or `false`, and disables or enables all IPv6 unicast forwarding on the router.

`plumbing`: this delimits a part of the router configuration used for the plumbing together of packet forwarding functionality. Multicast forwarding configuration must be part of this grouping.

`mfea4`: this delimits the part of the router configuration related to multicast forwarding of IPv4 packets.

The following multicast forwarding parameters can be configured:

- `disable`: this takes the value `true` or `false`, and disables or enables all IPv4 multicast forwarding on the router [3]. The default is `false`.
- `interface`: this specifies an interface to be used for multicast IPv4 forwarding. Each interface to be used for multicast forwarding needs to be explicitly listed.

  In addition to the normal network interfaces, a special-purpose interface called `register_vif` needs to be configured for PIM-SM (see Chapter 11) to be able to send register-encapsulated packets to the PIM Rendezvous Point. PIM-SM will not work correctly unless this is configured. The `register_vif` interface must be configured with a vif also called `register_vif`.
- `vif`: this specifies a vif to be used for multicast IPv4 forwarding. Each vif to be used for multicast forwarding needs to be explicitly listed.

  Each vif can take the following parameter:

  - `disable`: this takes the value `true` or `false`, and disables or enables multicast forwarding on this vif [4]. The default is `false`.

`traceoptions`: this directive delimits the configuration of debugging and tracing options for multicast forwarding.

- `flag`: this directive is used to specify which tracing options are enabled. Possible parameters are:
  - `all`: this directive specifies that all tracing options should be enabled. Possible parameters are:

[1] Note that prior to XORP Release-1.1, the `enable-unicast-forwarding4` flag was used instead to enable or disable the IPv4 forwarding.

[2] Note that prior to XORP Release-1.1, the `enable-unicast-forwarding6` flag was used instead to enable or disable the IPv6 forwarding.

[3] Note that prior to XORP Release-1.1, the `enable` flag was used instead of `disable`.

[4] Note that prior to XORP Release-1.1, the `enable` flag was used instead of `disable`.

`disable`: this takes the value `true` or `false`, and disables or enables tracing [5]. The default is `false`.

`mfea6`: this delimits the part of the router configuration related to multicast forwarding of IPv6 packets. The possible parameters are the same as for `mfea4`, but affect IPv6 multicast forwarding rather than IPv4.

4.2.2 Example Configurations

```
fea {
  unicast-forwarding4 {
    disable: false
  }
  unicast-forwarding6 {
    disable: true
  }
} plumbing {
  mfea4 {
    disable: false
    interface dc0 {
      vif dc0 {
        disable: false
      }
    }
    interface register_vif {
      vif register_vif {
        /* Note: this vif should be always enabled */
        disable: false
      }
    }
    traceoptions {
      flag all {
        disable: false
      }
    }
  }

  mfea6 {
    disable: false
    interface dc0 {
      vif dc0 {
        disable: false
      }
    }
    interface register_vif {
      vif register_vif {
        /* Note: this vif should be always enabled */
        disable: false
      }
    }
  }
}
```

The configuration above enables unicast IPv4 forwarding, but disables IPv6 unicast forwarding.

In addition, it enables multicast forwarding for IPv4 and IPv6 on interface/vif `dc0/dc0`, and enables the register vif for use by PIM-SM multicast routing.

[5] Note that prior to XORP Release-1.1, the `enable` flag was used instead of `disable`.

4.3 Monitoring the Forwarding Engine

The `show mfea dataflow` command can be used to display information about MFEA IPv4 dataflow filters:

```
Xorp> show mfea dataflow
Group                                  Source
224.0.1.20                             10.2.0.1
  Measured(Start|Packets|Bytes) Type Thresh(Interval|Packets|Bytes)   Remain
  1091667269.982158|0|?         <=     210.0|0|?                        202.434319
  1091667269.984406|?|0         >=     100.0|?|102400                   92.436567
```

Note that the above information is shown only if the filters are kept at user-space. If the filters are kept at kernel-space (*e.g.,* in case of UNIX system with advanced multicast API support), then currently xorpsh cannot be used to show the information. In that case, the appropriate system command should be used instead (e.g., the UNIX `netstat -gn` command).

The `show mfea interface` command can be used to display information about MFEA IPv4 interfaces:

```
Xorp> show mfea interface
Interface    State    Vif/PifIndex Addr            Flags
dc0          UP       0/6          10.4.0.1        MULTICAST BROADCAST KERN_UP
dc2          UP       1/8          10.3.0.2        MULTICAST BROADCAST KERN_UP
register_vif UP       2/6          10.4.0.1        PIM_REGISTER KERN_UP
```

The `show mfea interface address` command can be used to display information about MFEA IPv4 interface addresses:

```
Xorp> show mfea interface address
Interface    Addr            Subnet             Broadcast       P2Paddr
dc0          10.4.0.1        10.4.0.0/24        10.4.0.255      0.0.0.0
dc2          10.3.0.2        10.3.0.0/24        10.3.0.255      0.0.0.0
register_vif 10.4.0.1        10.4.0.1/32        10.4.0.1        0.0.0.0
```

The equivalent commands for IPv6 multicast forwarding are:

```
show mfea6 dataflow

show mfea6 interface

show mfea6 interface address
```

Chapter 5

Unicast Routing

5.1 An Overview of Unicast Routing

To forward packets, a router maintains a forwarding table which holds routes indicating which neighboring router a packet for a particular destination should be forwarded to. At the minimum, a route then consists of a destination *subnet* and a *nexthop*. The destination subnet is usually represented as a base IP address and a prefix-length in bits. For example, the subnet `128.16.64.0/24` has a prefix length of 24 bits, indicating that the first 24 bits of this address identify the network in question, and the last 8 bits identify hosts on this subnet. Thus a route for this subnet would be used to forward packets for addresses 128.16.64.0 to 128.16.64.255 inclusive. The nexthop can be the IP address of a neighboring router, or it might indicate that the route is for a subnet that is directly connected to this router.

IP routers perform *longest prefix match* forwarding. This means that a router might have more than one route that matches a destination address, and under such circumstances, it will use the route that has the longest prefix. For example, if a router has two routes:

- Subnet: 128.16.0.0/16, nexthop: 10.0.0.1
- Subnet: 128.16.64.0/24, nexthop: 10.0.0.2

A packet destined for 128.16.0.1 would match the first route only, and so would be forwarded to 10.0.0.1. However a packet destined for 128.16.64.1 would match both routes, and so would be forwarded to 10.0.0.2 because the second route has a longer prefix (24 is longer than 16).

To be useful, a router needs to populate its forwarding table. It does this in three ways:

- Routes for directly connected subnets are automatically entered into the forwarding table.
- Routes may be configured via the router's configuration file or command line interface. Such routes are known as *static routes*. Static routes will be discussed in Chapter 6.
- Routes may be learned from another router via a routing protocol. Such routes are known as *dynamic routes*.

5.1.1 Dynamic Routing

Many different routing protocols can supply dynamic routes. The dynamic routing protocols that are in most common use are:

- **Border Gateway Protocol (BGP)**. This is used for inter-domain routing.
- **Routing Information Protocol (RIP)**. This is probably the simplest intra-domain routing protocol, and is often used on small networks.
- **Open Shortest Path First (OSPF)**. Used for intra-domain routing, often on large ISP networks.
- **Integrated IS-IS**. Used for intra-domain routing, often on large ISP networks. Similar to OSPF.
- **IGRP**: Used for intra-domain routing, typically in small to medium sized networks. Cisco-proprietary.

Currently (April 2005), XORP supports BGP and RIP. We plan to implement OSPF and IS-IS shortly. Our BGP implementation is discussed in Chapter 8 and our RIP implementation is discussed in Chapter 7. In addition, there are also multicast routing protocols, which we will discuss in Chapter 9.

5.1.2 Administrative Distance

A router can run multiple routing protocols simultaneously. For example, we may use RIP to distribute routes within our network, and BGP to learn external routes. In some situations this can lead to a router learning the same route from more than one routing protocol. For example, we might learn the two routes:

- Subnet: 128.16.64.0/24, nexthop: 192.150.187.1, learned from BGP via an external peering. AS Path: `123 567 987`.
- Subnet: 128.16.64.0/24, nexthop: 10.0.0.2, learned from RIP with metric 13

The longest prefix match rule doesn't help us because the prefix lengths are the same, and the metric used for RIP is not directly comparable against the AS path length or any other attribute attached to a BGP route. How then do we decide which route to take?

A XORP router uses the concept of *administrative distance* to determine which route wins. This concept is the same as that used by Cisco routers. Basically each routing protocol has a configured "distance", and if a route is heard from two protocols, then the version with the smallest distance wins.

The built-in table of administative distances XORP uses is:

Directly connected subnets:	0
Static routes:	1
BGP, heard from external peer:	20
OSPF (when implemented):	110
IS-IS (when implemented):	115
RIP:	120
BGP, heard from internal peer:	200
FIB2MRIB routes (XORP-specific, in MRIB only):	254

Hence, in the example above, the route learned from BGP will be prefered.

Currently (April 2005), there is no way to modify these default administrative distances, but this capability will be added soon.

5.1.3 Route Redistribution

A common requirement is to redistribute routes between routing protocols. Some examples might be:

- When interconnecting some subnets that are statically routed with some subnets use RIP for dynamic routing. Rather that configure the static routes and additionally tell RIP to originate route advertisements for the same subnets, it is simpler and less error prone to configure the router to simply redistribute all the static routes into RIP.
- When a network uses RIP internally, and also uses BGP to peer with the rest of the Internet. One solution would be to configure BGP at the border routes to originate route advertisements for the internal subnets, but if a new subnet is added internally, then the border routers also need to be correctly modified. Instead we can simply configure the border routers to redistribute RIP routes into BGP.

Currently, XORP has an experimental policy mechanism that allows route redistribution. This feature will be enabled in the next release.

Chapter 6

Static Routes

6.1 Terminology and Concepts

A static route is a manually configured route. Static routes will not automatically change if a link or neighboring router fails. In general, static routes should only be used for very simple network topologies, or to override the behaviour of a dynamic routing protocol for a small number of routes.

Static routes can be configured for IPv4 and IPv6. Each route can be specified as to be used for unicast forwarding, or as part of the multicast topology used by multicast routing, or both.

The term *RIB* refers to the router's *Routing Information Base*. This is the collection of all routes the router has learned from its configuration or from its dynamic routing protocols. The RIB maintains separate collections of routes for IPv4 and IPv6. Within each of those collections, the router also maintains separate route tables for unicast routes and for multicast routes. Unicast routes will be used to determine the forwarding table used for unicast packet forwarding. Multicast routes do not directly determine the multicast forwarding table, but instead are used by multicast routing protocols such as PIM. PIM uses this to determine the RPF (Reverse-Path Forwarding) information[1] needed to route multicast control information that in turn sets up the multicast forwarding tree. The part of the *RIB* used to contain multicast topology information is called the *Multicast RIB* or *MRIB*.

[1]The RPF information represents the path back to a source.

6.2 Configuration of Static Routes

When a static route is specified, it is necessary to indicate not only the *destination subnet* and *next-hop* router, but also whether the route should be placed in the unicast RIB or in the MRIB or both.

6.2.1 Configuration Syntax

The syntax for defining static routes is shown below.

```
protocols {
  static {
    targetname: text
    disable: bool
    route4 IPv4-addr/int(0..32) {
      next-hop: IPv4-addr
      metric: uint
    }
    mrib-route4 IPv4-addr/int(0..32) {
      next-hop: IPv4-addr
      metric: uint
    }

    route6 IPv6-addr/int(0..128) {
      next-hop: IPv6-addr
      metric: uint
    }
    mrib-route6 IPv6-addr/int(0..128) {
      next-hop: IPv6-addr
      metric: uint
    }

    interface-route4 IPv4-addr/int(0..32) {
      next-hop-interface: text
      next-hop-vif: text
      metric: uint
    }
    mrib-interface-route4 IPv4-addr/int(0..32) {
      next-hop-interface: text
      next-hop-vif: text
      metric: uint
    }

    interface-route6 IPv6-addr/int(0..128) {
      next-hop-interface: text
      next-hop-vif: text
      metric: uint
    }
    mrib-interface-route6 IPv6-addr/int(0..128) {
      next-hop-interface: text
      next-hop-vif: text
      metric: uint
    }
  }
}
```

The configuration parameters are used as follows:

`protocols`: this delimits the configuration for all routing protocols in the XORP router configuration. It is mandatory that BGP configuration is under the `protocols` node in the configuration.

`static`: the delimits the part of the router configuration that is related to configuring static routes.

`targetname`: this is the name for this instance of static_routes. It defaults to "`static_routes`", and it is not recommended that this default is overridden under normal usage scenarios.

`disable`: this takes the value `true` or `false`, and determines whether any static routes are installed or not [2]. Setting it to `true` has the same effect as deleting the whole static routes configuration, but without losing what the old configuration actually was.

`route4`: this specifies an IPv4 unicast route to be installed in the RIB. The parameter is an IPv4 destination subnet expressed in the form *address/prefix-length*.

Each `route4`: specification takes the following mandatory attributes:

`next-hop`: this specifies the IPv4 address of the nexthop router towards the destination subnet [3].

`metric`: this specifies the routing metric or cost for this route. It is a non-negative integer. The metric for a static route is not directly used to decide which route to use, but may affect the choice of routes for protocols such as BGP and PIM-SM that indirectly use this information. For example, BGP uses the IGP metric to the nexthop to decide between alternative routes as part of its decision process. As with all routing metrics, lower values indicate better routes.

`mrib-route4`: this specifies an IPv4 multicast route to be installed in the Multicast RIB. The parameter is an IPv4 destination subnet expressed in the form *address/prefix-length*. This route will not directly affect forwarding, but will be used by multicast routing protocols such as PIM-SM to control how multicast trees are formed.

An `mrib-route4` specification takes the same attributes as a `route4` specification.

`route6`: this specifies an IPv6 unicast route to be installed in the RIB. The parameter is an IPv6 destination subnet expressed in the form *address/prefix-length*.

Each `route6`: specification takes the following mandatory attributes:

`next-hop`: this specifies the IPv6 address of the nexthop router towards the destination subnet.

`metric`: this specifies the routing metric or cost for this route. See `route4 metric` for details.

`mrib-route6`: this specifies an IPv4 multicast route to be installed in the Multicast RIB. The parameter is an IPv6 destination subnet expressed in the form *address/prefix-length*. This route will not directly affect forwarding, but will be used by multicast routing protocols such as PIM-SM to control how multicast trees are formed.

An `mrib-route6` specification takes the same attributes as a `route6` specification.

`interface-route4`: this specifies an IPv4 unicast route to be installed in the RIB. The parameter is an IPv4 destination subnet expressed in the form *address/prefix-length*. Typically, this specification will be used in wireless environment to install static routes where this router and next-hop router don't share the same subnet address on some (wireless) interface.

Each `interface-route4`: specification takes the following mandatory attributes:

`next-hop-interface`: this specifies the name of the nexthop interface towards the destination subnet.

[2]Note that prior to XORP Release-1.1, the `enable` flag was used instead of `disable`.

[3]Note that prior to the XORP Release-1.1, the `nexthop` attribute was used instead of `next-hop`.

`next-hop-vif`: this specifies the name of the nexthop vif towards the destination subnet.

`metric`: this specifies the routing metric or cost for this route. See `route4 metric` for details.

The `mrib-interface-route4` specification is same as the `interface-route4` specification, except that it is used to configure IPv4 routes that are to be installed in the Multicast RIB. The `interface-route6` and `mrib-interface-route6` specifications are also same, except that they are used to configure IPv6 routes (in the Unicast and the Multicast RIB respectively).

6.2.2 Example Configurations

```
protocols {
  static {
    route4 10.20.0.0/16 {
      next-hop: 10.10.10.20
      metric: 1
    }
    mrib-route4 10.20.0.0/16 {
      next-hop: 10.10.10.30
      metric: 1
    }

    route6 20:20:20:20::/64 {
      next-hop: 10:10:10:10:10:10:10:20
      metric: 1
    }
    mrib-route6 20:20:20:20::/64 {
      next-hop: 10:10:10:10:10:10:10:30
      metric: 1
    }
    interface-route4 10.30.0.0/16 {
      next-hop-interface: rl0
      next-hop-vif: rl0
      metric: 1
    }
    mrib-interface-route4 10.30.0.0/16 {
      next-hop-interface: rl1
      next-hop-vif: rl1
      metric: 1
    }

    interface-route6 30:30:30:30::/64 {
      next-hop-interface: rl0
      next-hop-vif: rl0
      metric: 1
    }
    mrib-interface-route6 30:30:30:30::/64 {
      next-hop-interface: rl1
      next-hop-vif: rl1
      metric: 1
    }
  }
}
```

6.3 Monitoring Static Routes

IPv4 unicast static routes can be displayed using the command `show route table ipv4 unicast static`:

```
Xorp> show route table ipv4 unicast static
Xorp> show route table ipv4 unicast static
Network 192.168.0.0/24
    Nexthop := 192.150.187.1
    Metric := 1 Protocol := static Interface := fxp0 Vif := fxp0
Network 192.168.1.0/24
    Nexthop := 192.150.187.2
    Metric := 1 Protocol := static Interface := fxp0 Vif := fxp0
Xorp>
```

The information shown for each route not only indicates the configured information (Network, Nexthop and Metric), but also the interface and vif via which this route will forward packets.

If the nexthop is not actually reachable, the route will not be shown by this command because there is not current interface or vif.

IPv6 unicast static routes can be displayed using the command `show route table ipv6 unicast static`.

The Multicast RIB static routes can be displayed using the commands `show route table ipv4 multicast static` and `show route table ipv6 multicast static` for IPv4 and IPv6 respectively.

Chapter 7

RIP and RIPng

7.1 Terminology and Concepts

The Routing Information Protocol (RIP) is the simplest unicast routing protocol in widespread use today. RIP is very simple, both in configuration and protocol design, so it is widely used in simple topologies. However, RIP does not scale well to larger networks, where OSPF or IS-IS might be more appropriate.

There have been two versions of the RIP protocol. RIP version 1 dates back to the early days of the Internet. It is now historic, primarily because it does not support classless addressing which is necessary in today's Internet. XORP does not support RIPv1.

RIP version 2 introduces a subnet mask, which allows classless addressing. XORP completely supports RIPv2, as specified in RFC 2453.

RIPng introduces IPv6 support. It is very similar to RIPv2, but for IPv6 instead of IPv4.

RIP is a distance vector protocol, which means that when a router receives a route from a neighbour, that route comes with a distance metric indicating the cost associated with reaching the destination via that neighbor. The router adds its metric for the link on which the route was received to the metric in the received route, and then compares the route against its current best path to that destination. If the metric is lower, or if there is no current route to the destination, then the new route wins, and is installed in the router's routing table. If the route is simply an update of the previous best route, then the stored metric is updated, and the route's deletion timer is restarted. Otherwise the route is ignored. Periodically, the router's routing table is sent to each of it's neighbors. Additionally, if a route changes, then the new route is sent to each neighbor.

On reason why RIP is not good for large networks is that in complex topologies it is rather slow to conclude that a route is no longer usable. This is because routers in a loop will learn a route from each other all the way around the loop, and so when a destination becomes unreachable, the routing change will have to propagate around the loop multiple times, increasing the metric each time until the metric reaches infinity, when the route is finally removed. RIP uses a low value of 15 as infinity to reduce the time it takes to remove old information.

A simple case of such a loop is two routers talking to each other. After a destination becomes unreachable, two routers may each believe the other has the best route. *Split horizon* is a scheme for avoiding problems caused by including routes in updates sent to the router from which they were learned. The *simple split* *horizon* scheme omits routes learned from one neighbor in updates sent to that neighbor. *Split horizon with poisoned reverse* includes such routes in updates, but sets their metrics to infinity. In general, it is advisable to use split-horizon with poisoned reverse when using RIP, as this significantly speeds convergence in many scenarios.

7.1.1 Standards Supported

XORP RIP complies with the following standards:

RFC 2453: RIP version 2.

RFC 2080: RIPng for IPv6.

7.2 Configuring RIP

To run RIP it is sufficient to specify the set of interfaces, vifs and addresses (`interface`, `vif` and `address`) on which RIP is enabled. Each `address` to be used by RIP must be explicitly configured, and typically a metric will also be configured.

In addition, to originate routes via RIP, it is necessary to use the `export` command to export routes from the router's routing table via RIP. At present, the configuration for route export is coarse-grain, allowing all routes originating from a particular routing "protocol" to be exported. For example, to export routes for directly connected interfaces, the directive "`export connected`" would be used.

Note that this mechanism for originating routes is temporary. In the next release it will be replaced by a more generic policy framework. Unfortunately, the same keyword `export` will be used but with different syntax.

7.2.1 Configuration Syntax

```
protocols {
  rip {
    targetname text
    export text {
      metric: int
      tag: int
    }
    interface text {
      vif text {
        address IPv4 {
          metric: uint
          horizon: text
          disable: bool
          passive: bool
          accept-non-rip-requests: bool
          accept-default-route: bool
          route-expiry-secs: uint
          route-deletion-secs: uint
          triggered-update-min-secs: uint
          triggered-update-max-secs: uint
          table-announce-min-secs: uint
          table-announce-max-secs: uint
          table-request-secs: uint
          interpacket-delay-msecs: uint
          authentication {
            type: text
            password: text
          }
        }
      }
    }
  }
}
```

`protocols`: this delimits the configuration for all routing protocols in the XORP router configuration. It is mandatory that RIP configuration is under the `protocols` node in the configuration.

`rip`: this delimits the RIP configuration part of the XORP router configuration.

`targetname`: this is the name for this instance of RIP. It defaults to "`rip`", and it is not recommended that this default is overridden under normal usage scenarios.

`export`: this directive specifies an export rule. Routes matching the export rule will be exported from the routing table via RIP to the outside world. The `export` directive takes a parameter indicating which routes are to be exported via RIP. Valid values are "`connected`" indicating directly connected subnets should be advertised via RIP, "`static`" indicating configured static routes should be re-advertised via RIP, and "`bgp`" indicating all BGP routes should be re-advertised via RIP. Re-advertising BGP routes is not advised unless the router has only a *very* limited BGP routing table.

Each export statement can take the following parameters:

`metric`: this specifies the metric RIP should advertise for these exported subnets. It takes an integer value between 1 and 15. Note that 15 is regarded as infinity as far as RIP is concerned. The sum of all the metrics across the entire RIP domain should be less than 15.

`tag`: this specifies the RIP tag should should advertise for these exported subnets. Tags can be used to distinguish between routes from different origins when doing policy filtering at routers receiving these routes. If specified, the tag must be an integer between 0 and 65535.

`interface`: this specifies a network interface that should be used by RIP for routing. See Chapter 3 for details of interfaces. The interface must be configured in the `interfaces` part of the router configuration.

Each interface can have multiple vifs configured:

`vif`: this specifies a vif that should be used by RIP for routing. See Chapter 3 for details of vifs.

`address`: this specifies an IPv4 address that should be used by RIP for routing. RIP will peer with other routers on this `interface/vif` using this `address`. The address must be a valid configured address for this vif.

The parameters that can be specified for each address are:

`metric`: this specifies the metric or cost associated with routes received on this vif/address. The metric is added to the cost in routes received before deciding between best routes to the same destination subnet. `metric` should be an integer between 1 and 15. Note that 15 is regarded as infinity as far as RIP is concerned. The sum of all the metrics across the entire RIP domain should be less than 15.

`horizon`: this specifies how RIP deals with eliminating routes quickly after a path has failed. Possible values are "`split-horizon-poison-reverse`", "`split-horizon`", and "`none`". The default is `split-horizon-poison-reverse` and under normal circumstances should be left unchanged.

`disable`: this takes the value `true` or `false`, and determines whether RIP will exchange routes via this vif/address [1]. Setting this to `true` allows routes received via an address to be temporarily removed without deleting the configuration. The default is `false`.

`passive`: this takes the value `true` or `false`, and determines whether RIP runs in passive mode on this address. In passive mode, RIP will accept routes received on this address, but will not advertise any routes to neighbors via this address. The default is `false`.

`accept-non-rip-requests`: this takes the value `true` or `false`. Normal RIPv2 requests for routing updates are multicast to all neighbors and sourced from the RIP port. However for monitoring purposes RIP also allows requests to be unicast, and then they can be sourced from non-RIP ports. When this option is `true`, RIP will accept RIP requests from any UDP port. The default is `true`.

[1] Note that prior to XORP Release-1.1, the `enable` flag was used instead of `disable`.

`accept-default-route`: this takes the value `true` or `false`, and indicates whether RIP should accept a default route if it receives one from a RIP neighbor. The default is `false`.

`route-expiry-secs`: If no periodic or triggered update of a route from this neighbor has been received for this time interval, the route is considered to have expired. The default is 180 seconds, and should not normally need to be changed.

`route-deletion-secs`: After a route has expired (the route has an infinite metric), a router must keep a copy of it for a certain time so it can have a reasonably confidence that it has told its neighbors that the route has expired. This time interval determines how long the router maintains expired routes after their metric has reached infinity. The default is 120 seconds, and should not normally need to be changed.

`triggered-update-min-secs`: When a router receives a modified route from a neighbor, it does not have to wait until the next periodic update to tell the other neighbors, but instead sends a triggered update. After a triggered update is sent, a timer is set for a random interval between `triggered-update-min-secs` and `triggered-update-max-secs`. If other changes occur that would trigger updates before the timer expires, a single update is triggered when the timer expires. The default value of `triggered-update-min-secs` is 1 second, and should not normally need to be changed.

`triggered-update-max-secs`: See `triggered-update-min-secs` for details. The default is 5 seconds, and should not normally need to be changed.

`table-announce-min-secs`: A RIP router will typically tell its neighbours its entire routing table every 30 seconds. To avoid self-synchronization of routing updates, the precise time interval between telling each neighbor about routing updates is randomly jittered, with the delay chosen uniformly at random between `table-announce-min-secs` and `table-announce-max-secs`. The default for `table-announce-min-secs` is 25 seconds, and should not normally need to be changed.

`table-announce-max-secs`: See `table-announce-min-secs` for details. The default is 35 seconds, and should not normally need to be changed.

`table-request-secs`: When a RIP router has no neighbors on a vif/address, it may periodically send a request for a route update in case a neighbor appears. This timer determines how often such a request is re-sent. The default value is 1 second.

`interpacket-delay-msecs`: This specifies the default delay between back-to-back RIP packets when an update is sent that requires multiple packets to be sent. The default is 50 milliseconds, and should not normally need to be changed.

`authentication`: This directive specifies the authentication mechanism used to authorise RIP updates sent and received via this vif/address.

The two parameters used for authentication are:

`type`: this specifies the type of authentication to be used. Valid values are “`none`”, “`plaintext`”, and “`md5`”. Plaintext authentication includes an unencrypted password in every RIP update packet. MD5 authentication includes a keyed MD5 digest in every RIP update packet, and is rather more secure than plaintext.

`password`: this specifies the password used for plaintext or md5 authentication on this vif/address.

7.3 Configuring RIPng

The configuration for RIP is basically the same as for RIP, with two exceptions:

- The addresses are IPv6 addresses with RIPng whereas they are IPv4 addresses with RIPv2.
- The `authentication` directive is not available in RIPng, because RFC 2081 does not specify authentication for RIPng.

7.3.1 Example Configurations

```
protocols {
  rip {
  /* Redistribute routes for connected interfaces */
    export connected {
      metric: 0
      tag: 0
    }
  /* Redistribute static routes */
    export static {
      metric: 1
      tag: 0
    }
  /* Run on specified network interface addresses */
    interface dc0 {
      vif dc0 {
        address 10.10.10.10 {
          disable: false
        }
      }
    }
  }
}
```

In the above configuration, RIP is configured to export routes for directly connected subnets and for routes that are statically configured. The RIP metric advertised is configured to be 0 for connected subnets and 1 for static routes.

RIP is configured on only one interface/vif (`dc0/dc0`), with address 10.10.10.10. This router will send and receive routes from any RIP neighbors that it discovers on that vif/address.

7.4 Monitoring RIP

RIP routes can be monitored using the operational mode command:
`show route table ipv4 unicast rip`.
For each subnet, the nexthop router, the RIP metric, and the interface/vif to reach the nexthop route are shown.

```
Xorp> show route table ipv4 unicast rip
Network 172.16.0.0/24
    Nexthop := 172.16.0.1
    Metric := 1 Protocol := rip Interface := dc0 Vif := dc0
Network 172.16.1.0/24
    Nexthop := 172.16.1.1
    Metric := 1 Protocol := rip Interface := dc1 Vif := dc1
Network 172.16.2.0/24
    Nexthop := 172.16.2.1
    Metric := 1 Protocol := rip Interface := dc2 Vif := dc2
Network 172.16.3.0/24
    Nexthop := 172.16.3.1
    Metric := 1 Protocol := rip Interface := dc3 Vif := dc3
Network 192.150.187.0/25
    Nexthop := 192.150.187.112
    Metric := 1 Protocol := rip Interface := fxp0 Vif := fxp0
```

The operational command for monitoring the IPv6 unicast routes is `show route table ipv6 unicast rip`. The operational commands for monitoring the MRIB routes are `show route table ipv4 multicast rip` and `show route table ipv6 multicast rip` for IPv4 and IPv6 respectively.

Chapter 8

BGP

8.1 BGP Terminology and Concepts

BGP is the Border Gateway Protocol, which is the principal inter-domain routing protocol in the Internet. BGP version 4 is specified in RFC 1771. However, RFC 1771 is shortly to be replaced by a much improved newer RFC, and XORP BGP is compliant with the new RFC. Earlier versions of BGP are now considered historic. XORP implements what is known as BGP4+. This is the core BGP-4 protocol, plus the multi-protocol extensions needed to route IPv6 traffic and to provide separate topology information for multicast routing protocols to that used for unicast routing.

A complete description of BGP is outside the scope of this manual, but we will mention a few of the main concepts.

8.1.1 Key BGP Concepts

The main concept used in BGP is that of the Autonomous System, or AS for short. An AS corresponds to a routing domain that is under one administrative authority, and which implements its own routing policies. BGP is used in two different ways:

- EBGP is used to exchange routing information between routers that are in different ASes.
- IBGP is used to exchange routing information between routers that are in the same AS. Typically these routes were originally learned from EBGP.

Each BGP route carries with it an AS Path, which essentially records the autonomous systems through which the route has passed between the AS where the route was originally advertised and the current AS. When a BGP router passes a route to a router in a neighboring AS, it prepends its own AS number to the AS path. The AS path is used to prevent routes from looping, and also can be used in policy filters to decide whether or not to accept a route.

When a route reaches a router over an EBGP connection, the router first decides if this is the best path to the destination, based on a complex decision process and local policy configuration. If the route is the best path, the route is passed on to all the other BGP routers in the same domain using IBGP connections, as well as on to all the EBGP peers (as allowed by policy).

When a router receives a route from an IBGP peer, if the router decides this route is the best route to the destination, then it will pass the route on to its EBGP peers, but it will not normally pass the route onto another IBGP peer. This prevents routing information looping within the AS, but it means that by default every BGP router in a domain must be peered with every other BGP router in the domain.

Of course such a full mesh of configured BGP peerings does not scale well to large domains, so two techniques can be used to improve scaling:

- Confederations.
- Route Reflectors.

As of the 1.0 release, XORP BGP does not support confederations, and does not support being used as a route reflector, although it can be used as a route reflector client. This capability will be added soon.

BGP peerings are conducted over TCP connections which must be manually configured. A connection is an IBGP peering if both routers are configured to be in the same AS; otherwise it is an EBGP peering.

Routers typically have multiple IP addresses, with at least one for each interface, and often an additional routable IP address associated with the loopback interface[1]. When configuring an IBGP connection, it is good practice to set up the peering to be between the IP addresses on the loopback interfaces. This makes the connection independent of the state of any particular interface. However, most EBGP peerings will be configured using the IP address of the router that is directly connected to the EBGP peer router. Thus if the interface to that peer goes down, the peering session will also go down, causing the routing to correctly fail over to an alternative path.

8.2 Standards

XORP BGP complies with the following standards:

draft-ietf-idr-bgp4-22.txt: BGP-4 Specification (obsoletes RFC 1771)

RFC 1657: Definitions of Managed Objects for the Fourth Version of the Border Gateway Protocol (BGP-4) using SMIv2.

RFC 2858: Multiprotocol Extensions for BGP-4.

RFC 2545: Use of BGP-4 Multiprotocol Extensions for IPv6 Inter-Domain Routing.

We also have limited support for:

RFC 1997: BGP Communities Attribute.

[1]Note: 127.0.0.1 is *not* routable.

8.3 Configuring BGP

8.3.1 Configuration Syntax

The configuration syntax for XORP BGP is given below.

```
protocols {
  bgp {
    targetname: text
    bgp-id: IPv4
    local-as: int(1..65535)

    peer text {
      local-ip: IPv4
      as: int(1..65535)
      next-hop: IPv4
      local-port: int(1..65535)
      peer-port: int(1..65535)
      holdtime: uint
      disable: bool
      ipv4-unicast: bool
      ipv4-multicast: bool
      ipv6-unicast: bool
      ipv6-multicast: bool
    }

    network4 IPv4/int(1..32) {
      next-hop: IPv4
      unicast: bool
      multicast: bool
    }

    network6 IPv6/int(1..128) {
      next-hop: IPv6
      unicast: bool
      multicast: bool
    }
}
```

The configuration parameters are used as follows:

`protocols`: this delimits the configuration for all routing protocols in the XORP router configuration. It is mandatory that BGP configuration is under the `protocols` node in the configuration.

`bgp`: this delimits the BGP configuration part of the XORP router configuration.

`targetname`: this is the name for this instance of BGP. It defaults to "`bgp`", and it is not recommended that this default is overridden under normal usage scenarios.

`bgp-id`: this is the BGP identifier for the BGP instance on this router. It is typically set to one of the router's IP addresses, and it is normally required that this is globally unique. The required format of the BGP ID is a dotted-decimal IPv4 address, as mandated by the BGP specification. This is required even if the router only supports IPv6 forwarding.

`local-as`: this is the autonomous system number for the AS in which this router resides. Any peers of this router must be configured to know this AS number - if there is a mismatch, a peering will not be established. It is a 16-bit integer.

`peer`: this delimits the configuration of a BGP peering association with another router. Most BGP routers will have multiple peerings configured. The `peer` directive takes a parameter which is the peer identifier for the peer router. This peer identifier should normally be the IPv4 unicast address of the router we are peering with. The syntax allows it to be the domain names of the peer router for convenience, but this is *not* recommended in production settings.

For IBGP peerings the peer identifier will normally be an IP address bound to the router's loopback address, so it is not associated with a specific interface, meaning that the peering will not go down if a single internal interface fails.

For EBGP peerings, the peer identifier will normally be the IP address of the peer router on the interface over which we wish to exchange traffic, so that if the interface goes down, the peering will drop.

For each configured `peer`, the following configuration options can be specified:

`local-ip`: This is the IP address of this router that we will use for BGP connections to this peer. It is mandatory to specify, and must be the same as the IP address configured on the peer router for this peering.

`as`: this gives the AS number of this peer. This must match the AS number that the peer itself advertises to us, or the BGP peering will not be established. It is a 16-bit integer, and is mandatory to specify.

`next-hop`: this is the IPv4 address that will be sent as the nexthop router address in routes that we send to this peer. Typically this is only specified for EBGP peerings.

`next-hop6`: this is the IPv6 address that will be sent as the nexthop router address in routes that we send to this peer. Typically this is only specified for EBGP peerings.

`local-port`: by default, BGP establishes its BGP connections over a TCP connection between port 179 on the local router and port 179 on the remote router. The local port for this peering can be changed by modifying this attribute. This must be the same as the corresponding `remote-port` on the remote peer router or a connection will not be established.

`peer-port`: The port for this peering on the remote router can be changed by modifying this attribute. See also: `local-port`.

`holdtime`: This is the holdtime BGP should use when negotiating the connection with this peer. If no message is received from a BGP peer during the negotiated holdtime, the peering will be shut down.

`disable`: This takes the value `true` or `false`, and indicates whether the peering is currently disabled. This allows a peering to be taken down temporarily without removing the configuration [2].

`ipv4-unicast`: This takes the value `true` or `false`, and specifies whether BGP should negotiate multiprotocol support with this peer to allow IPv4 unicast routes to be exchanged. It is enabled by default.

`ipv4-multicast`: This takes the value `true` or `false`, and specifies whether BGP should negotiate multiprotocol support with this peer to allow separate routes to be used for IPv4 unicast and IPv4 multicast. Normally this would only be enabled if PIM-SM multicast routing is running on the router.

`ipv6-unicast`: This takes the value `true` or `false`, and specifies whether BGP should negotiate multiprotocol support with this peer to allow IPv6 unicast routes to be exchanged.

[2] Note that prior to XORP Release-1.1, the `enable` flag was used instead of `disable`.

`ipv6-multicast`: This takes the value `true` or `false`, and specifies whether BGP should negotiate multiprotocol support with this peer to allow IPv6 multicast routes to be exchanged separately from IPv6 unicast routes. It is possible to enable IPv6 multicast support without enabling IPv6 unicast support.

`network4`: this specifies an IPv4 route to be advertised into the BGP routing, originating from this router. The parameter is an IPv4 subnet in the format *address/prefix-length*. For example `10.0.0.0/24`.

`next-hop`: this is the IPv4 address of the next-hop router to be used in BGP advertisements for this route. Typically it will be one of the router's IP addresses.

`unicast`: this takes the value `true` or `false` and specifies whether this route should be advertised to be used to route unicast packets.

`multicast`: this takes the value `true` or `false` and specifies whether this route should be advertised to be used to route multicast packets. Either `unicast` or `multicast` or both must be true.

`network6`: this specifies an IPv6 route to be advertised into the BGP routing, originating from this router. The parameter is an IPv6 subnet in the format *address/prefix-length*. For example `10:0:0:10::/64`.

8.3.2 Example Configurations

```
protocols {
  bgp {
    bgp-id: 128.16.32.1
    local-as: 45678

    peer 192.168.150.1 {
      local-ip: 128.16.64.4
      as: 34567
      next-hop: 128.16.64.4
      holdtime: 120

      /* IPv4 unicast is enabled by default */
      ipv4-unicast: true

      /* Optionally enable other AFI/SAFI combinations */
      ipv4-multicast: true
      ipv6-unicast: true
      ipv6-multicast: true
    }

    /* Originate IPv4 Routes */
    network4 128.16.16/24 {
      next-hop: 128.16.64.1
      unicast: true
      multicast: true
    }

    /* Originate IPv6 Routes */
    network6 10:10:10:10::/64 {
      next-hop: 10:10:10:10:10:10:10:10
      unicast: true
      multicast: false
    }
  }
}
```

This configuration is from a BGP router in AS 45678. The router has a BGP identifier of 128.16.32.1, which will normally be one of the router's IP addresses.

This router has only one BGP peering configured, with a peer on IP address 192.168.150.1. This peering is an EBGP connection because the peer is in a different AS (34567). This router's IP address used for this peering is 128.16.64.4, and the router is also configured to set the next hop router field in routes it advertises to the peer to be 128.16.64.4. Setting local-ip and next-hop to be the same is common for EBGP connections. The holdtime for the peering is configured to be 120 seconds, but the precise value of the holdtime actually used depends on negotiation with the peer. In addition to IPv4 unicast routing, which is enabled by default, this peering is configured to allow the sending an receiving of IPv4 multicast routes and IPv6 unicast routes.

This router is also configured to *originate* routing advertisements for two subnets. These subnets might be directly connected, or might be reachable via IGP routing.

The first advertisement this router originates is for subnet 128.16.16/24, reachable via both unicast and multicast. The nexthop specified in 128.16.64.1, and this must be reachable via other routes in the routing table, or this advertisement will not be made. If this router had any IBGP peerings, then the BGP route advertised to those peers would indicate that 128.16.16/24 was reachable via next hop 128.16.64.1. However in this case the only peering is an EBGP peering, and the next hop in *all* routes sent to that peer is set to 128.16.64.4 according to the `nexthop` directive for the peering.

The second advertisement is for an IPv6 route, configured to be usable only by IPv6 unicast traffic.

8.4 Monitoring BGP

On a router running BGP, the BGP routing state can be displayed using the `show bgp` operational-mode command. Information is available about the status of BGP peerings and about the routes received and used. In the 1.0 release, the set of commands is fairly limited, and will be increased in future releases to provide better ways to display subsets of this information.

As always, command completion using <TAB> or ? will display the available sub-commands and parameters:

```
Xorp> show bgp ?
Possible completions:
  <[Enter]>       Execute this command
  peers           Show BGP peers info
  routes          Print BGP routes
  |               Pipe through a command
```

The `show bgp peers` command will display information about the BGP peerings that have been configured. It supports the optional parameter `detail` to give a lot more information:

```
Xorp> show bgp peers ?
Possible completions:
  <[Enter]>       Execute this command
  detail          Show detailed BGP peers info
  |               Pipe through a command
```

By itself, `show bgp peers` provides a short list of the peerings that are configured, irrespective of whether the peering is in established state or not:

```
Xorp> show bgp peers
Peer 1: local 192.150.187.112/179 remote 69.110.224.158/179
Peer 2: local 192.150.187.112/179 remote 192.150.187.2/179
Peer 3: local 192.150.187.112/179 remote 192.150.187.78/179
Peer 4: local 192.150.187.112/179 remote 192.150.187.79/179
Peer 5: local 192.150.187.112/179 remote 192.150.187.109/179
```

The command `show bgp peers detail` will give a large amount of information about all the peerings:

```
Xorp> show bgp peers detail
Peer 1: local 192.150.187.112/179 remote 69.110.224.158/179
  Peer ID: none
  Peer State: ACTIVE
  Admin State: START
  Negotiated BGP Version: n/a
  Peer AS Number: 65014
  Updates Received: 0, Updates Sent: 0
  Messages Received: 0, Messages Sent: 0
  Time since last received update: n/a
  Number of transitions to ESTABLISHED: 0
  Time since last in ESTABLISHED state: n/a
  Retry Interval: 120 seconds
  Hold Time: n/a, Keep Alive Time: n/a
  Configured Hold Time: 120 seconds, Configured Keep Alive Time: 40 seconds
  Minimum AS Origination Interval: 0 seconds
  Minimum Route Advertisement Interval: 0 seconds

Peer 2: local 192.150.187.112/179 remote 192.150.187.2/179
  Peer ID: 192.150.187.2
  Peer State: ESTABLISHED
  Admin State: START
  Negotiated BGP Version: 4
  Peer AS Number: 64999
  Updates Received: 52786, Updates Sent: 28
  Messages Received: 52949, Messages Sent: 189
  Time since last received update: 2 seconds
  Number of transitions to ESTABLISHED: 17
  Time since last entering ESTABLISHED state: 6478 seconds
  Retry Interval: 120 seconds
  Hold Time: 120 seconds, Keep Alive Time: 40 seconds
  Configured Hold Time: 120 seconds, Configured Keep Alive Time: 40 seconds
  Minimum AS Origination Interval: 0 seconds
  Minimum Route Advertisement Interval: 0 seconds
```

The most important piece of information is typically whether or not the peering is in ESTABLISHED state, indicating that the peering is up and capable of exchanging routes. ACTIVE state means that the peering is configured to be up on this router, but for some reason the peering is not currently up. Typically this is because the remote peer is unreachable, or because no BGP instance is running on the remote peer.

The `show bgp routes` command displays the routes received by BGP from its peers. On a router with a full BGP routing table (140000 routes as of July 2004) this command will produce a large amount of output:

```
Xorp> show bgp routes
Status Codes: * valid route, > best route
Origin Codes: i IGP, e EGP, ? incomplete

   Prefix           Nexthop          Peer            AS Path
   ------           -------          ----            -------
*> 3.0.0.0/8        192.150.187.2    192.150.187.2   16694 25 2152 3356 7018 80 i
*> 4.17.225.0/24    192.150.187.2    192.150.187.2   16694 25 2152 11423 209 701 11853 6496 i
*> 4.17.226.0/23    192.150.187.2    192.150.187.2   16694 25 2152 11423 209 701 11853 6496 i
*> 4.17.251.0/24    192.150.187.2    192.150.187.2   16694 25 2152 11423 209 701 11853 6496 i
*> 4.17.252.0/23    192.150.187.2    192.150.187.2   16694 25 2152 11423 209 701 11853 6496 i
*> 4.21.252.0/23    192.150.187.2    192.150.187.2   16694 25 2152 11423 209 701 6389 8063 19198 i
*> 4.23.180.0/24    192.150.187.2    192.150.187.2   16694 25 2152 11423 209 3561 6128 30576 i
*> 4.36.200.0/21    192.150.187.2    192.150.187.2   16694 25 2152 174 3561 14742 11854 14135 i
*> 4.78.0.0/21      192.150.187.2    192.150.187.2   16694 25 2152 11423 209 3561 6347 23071 22938 i
*> 4.78.32.0/21     192.150.187.2    192.150.187.2   16694 25 2152 174 3491 29748 i
*> 4.0.0.0/8        192.150.187.2    192.150.187.2   16694 25 2152 3356 i
...
```

The format of the output is one route per line. On each line:

- A status code is displayed, showing whether the route is valid, and whether it was the best BGP route this router has received. A route is valid if the nexthop is reachable and it isn't filtered by the inbound BGP filters.
- The network prefix for which the route applies is listed in the form `4.17.226.0/23`. This indicates the base address for the network (address `4.17.226.0`), and the prefix length (`23` bits). Thus this route applies for addresses `4.17.226.0` to `4.17.227.255` inclusive.
- The nexthop is the IP address of the intermediate router towards which packet destined for the network prefix should be sent. In this example all the displayed routes have the same nexthop.
- The peer is the IP address of the BGP router which sent us this route. The nexthop and the peer need not the the same (they often aren't with IBGP peerings for example) but in all the routes in this example they are the same.
- The AS path is listed next. This lists the AS numbers of the autonomous systems that the route has traversed to reach our router. The AS at the left end of the path is the one nearest to our router and the one at the right end of the path is usually the AS number of the route's originator.
- Finally, whether the route's origin is from an IGP (`i`), from EGP (`e`, mostly obsolete), or incomplete (`?`) is listed.

8.4.1 BGP MIB

XORP includes SNMP support for BGP, though the BGP-4 MIB defined in RFC 1657.

Chapter 9

Multicast Routing

9.1 An Overview of Multicast Routing

IP Multicast is a technology that allows one-to-many and many-to-many distribution of data on the Internet. Senders send their data to a multicast IP destination address, and receives express an interest in receiving traffic destined for such an address. The network then figures out how to get the data from senders to receivers.

If both the sender and receiver for a multicast group are on the same local broadcast subnet, then the routers do not need to be involved in the process, and communication can take place directly. If, however, the sender and receiver are on different subnets, then a multicast routing protocol needs to be involved in setting up multicast forwarding state on the tree between the sender and the receivers.

9.1.1 Multicast Routing

Broadly speaking, there are two different types of multicast routing protocols:

- Dense-mode protocols, where traffic from a new multicast source is delivered to all possible receivers, and then subnets where there are no members request to be pruned from the distribution tree.
- Sparse-mode protocols, where explicit control messages are used to ensure that traffic is only delivered to the subnets where there are receivers that requested to receive it.

Examples of dense-mode protocols are *DVMRP* and *PIM Dense Mode*. Examples of sparse-mode protocols are PIM Sparse Mode, CBT, and MOSPF. Most of these protocols are largely historic at this time, with the exception of PIM Sparse Mode (PIM-SM) and PIM Dense Mode (PIM-DM), and even PIM-DM is not very widely used.

In addition to the routing protocols used to set up forwarding state between subnets, a way is needed for the routers to discover that there are local receivers on a directly attached subnet. For IPv4 this role is served by the Internet Group Management Protocol (IGMP) and for IPv6 this role is served by the Multicast Listener Discovery protocol (MLD).

9.1.2 Service Models: ASM vs SSM

There are two different models for IP multicast:

- Any Source Multicast (ASM), in which a receiver joins a multicast group, and receives traffic from any senders that send to that group.
- Source-Specific Multicast (SSM), in which a receiver explicitly joins to a (source, group) pairing.

Traditionally IP multicast used the ASM model, but problems deploying inter-domain IP multicast resulted in the much simpler SSM model being proposed. In the future it is likely that ASM will continue to be used within intranets and enterprises, but SSM will be used when multicast is used inter-domain. The two models are compatible, and PIM-SM can be used as a multicast routing protocol for both. The principal difference is that ASM only requires IGMPv2 or MLDv1, whereas SSM requires IGMPv3 or MLDv2 to permit the receivers to specify the address of the sending host.

9.1.3 Multicast Addresses

For IPv4, multicast addresses are in the range 224.0.0.0 to 239.255.255.255 inclusive. Addresses within 224.0.0.0/24 are considered link-local and should not be forwarded between subnets. Addresses within 232.0.0.0/8 are reserved for SSM usage. Addresses in 239.0.0.0/8 are ASM addresses defined for varying sizes of limited scope.

IPv6 multicast addresses are a little more complex. IPv6 multicast addresses start with the prefix `ff`, and have the following format:

```
|   8    |  4 |  4 |                  112 bits                         |
+------ -+----+----+---------------------------------------------------+
|11111111|flgs|scop|                  group ID                         |
+--------+----+----+---------------------------------------------------+
```

- `11111111` (`ff` in hexadecimal) at the start of the address identifies the address as being a multicast address.
- *flgs* is a set of 4 flags:

  ```
  +-+-+-+-+
  |0|0|0|T|
  +-+-+-+-+
  ```

  The high-order 3 flags are reserved, and must be initialized to 0.

  T = 0 indicates a permanently-assigned ("well-known") multicast address, assigned by the global internet numbering authority.

  T = 1 indicates a non-permanently-assigned ("transient") multicast address.
- *scop* is a 4-bit multicast scope value used to limit the scope of the multicast group. The values in hex are:

  `1` node-local scope

`2` link-local scope

`5` site-local scope

`8` organization-local scope

`E` global scope

- *group ID* identifies the multicast group, either permanent or transient, within the given scope.

RFC 2373 gives more details about IPv6 multicast addresses.

9.2 Supported Protocols

XORP supports the following multicast protocols:

- PIM Sparse Mode for both ASM and SSM multicast routing for IPv4.
- PIM Sparse Mode for both ASM and SSM multicast routing for IPv6.
- IGMPv1 and IGMPv2 for IPv4 local multicast membership.
- MLDv1 for IPv6 local multicast membership.

XORP does not yet support IGMPv3 and MLDv2, but plans to do so soon.

Chapter 10

IGMP and MLD

10.1 Terminology and Concepts

When a receiver joins a multicast group, the multicast routers serving that receiver's subnet need to know that the receiver has joined so that they can arrange for multicast traffic destined for that group to reach this subnet. The Internet Group Management Protocol (IGMP) is a link-local protocol for IPv4 that communicates this information between receivers and routers. The same role for IPv6 is performed by the Multicast Listener Discovery protocol (MLD).

The basic IGMP mechanism works as follows. When a multicast receiver joins a multicast group it multicasts an IGMP Join message onto the subnet on which it is joining. The local routers receive this join, and cause multicast traffic destined for the group to reach this subnet. Periodically one of the local routers sends a IGMP Query message onto the subnet. If there are multiple multicast routers on the subnet, then one of them is elected as the sole querier for that subnet. In response to an IGMP query, receivers respond by refreshing their IGMP Join. If the join is not refreshed in response to queries, then the state is removed, and multicast traffic for this group ceases to reach this subnet.

There are three different versions of IGMP:

- IGMP version 1 functions as described above.
- IGMP version 2 adds support for IGMP Leave messages to allow fast leave from a multicast group.
- IGMP version 3 adds support for source include and exclude lists, to allow a receiver in indicate that it only wants to hear traffic from certain sources, or not receive traffic from certain sources.

Currently XORP supports IGMPv1 and IGMPv2.

MLD for IPv6 functions in basically the same way as IGMP. The functionality of MLDv1 corresponds with that of IGMPv2, and the functionality of MLDv2 corresponds with that of IGMPv3.

Currently XORP supports MLDv1.

10.2 Standards

XORP complies with the following standards for multicast group membership:

RFC 2236: Internet Group Management Protocol, Version 2

RFC 1112: Host extensions for IP multicasting.

RFC 2710: Multicast Listener Discovery (MLD) for IPv6.

10.3 Configuring IGMP and MLD

IGMP and MLD only require the interfaces/vifs to be configured that are intended to have multicast listeners.

10.3.1 Configuration Syntax

```
protocols {
  igmp {
    targetname: text
    disable: bool
    interface text {
      vif text {
        disable: bool
      }
    }
    traceoptions {
      flag all {
        disable: bool
      }
    }
  }
}

protocols {
  mld {
    targetname: text
    disable: bool
    interface text {
      vif text {
        disable: bool
      }
    }
    traceoptions {
      flag all {
        disable: bool
      }
    }
  }
}
```

`protocols`: this delimits the configuration for all routing protocols in the XORP router configuration. It is mandatory that IGMP configuration is under the `protocols` node in the configuration.

`igmp`: this delimits the IGMP configuration part of the XORP router configuration.

`targetname`: this is the name for this instance of IGMP. It defaults to "`IGMP`", and it is not recommended that this default is overridden under normal usage scenarios.

`disable`: this takes the value `true` or `false`, and determines whether IGMP as a whole is enabled on this router [1]. The default value is `false`.

`interface`: this specifies an interface to be monitored by IGMP for the presence of multicast receivers. Each interface to be monitored by IGMP needs to be explicitly listed. The value is the name of an interface that has been configured in the `interfaces` section of the router configuration (see Chapter 3).

For each interface, one or more VIFs must be specified:

- `vif`: this specifies a vif to be monitored by IGMP for the presence of multicast receivers. Each vif to be monitored by IGMP needs to be explicitly listed. The value is the name of a vif that has been configured in the `interfaces` section of the router configuration (see Chapter 3).

  Each vif takes the following optional parameter:

  - `disable`: this takes the value `true` or `false`, and determines whether IGMP is disabled on this vif [2]. The default value is `false`.

`traceoptions`: this directive delimits the configuration of debugging and tracing options for IGMP.

- `flag`: this directive is used to specify which tracing options are enabled. Possible parameters are:
  - `all`: this directive specifies that all tracing options should be enabled. Possible parameters are:
    - `disable`: this takes the value `true` or `false`, and disables or enables tracing [3]. The default is `false`.

The configuration parameters for MLD are identical to those for IGMP, except that they are delimited by an `mld` directive rather than an `igmp` directive.

[1] Note that prior to XORP Release-1.1, the `enable` flag was used instead of `disable`.
[2] Note that prior to XORP Release-1.1, the `enable` flag was used instead of `disable`.
[3] Note that prior to XORP Release-1.1, the `enable` flag was used instead of `disable`.

10.3.2 Example Configurations

```
protocols {
  igmp {
    interface dc0 {
      vif dc0
    }
    interface dc1 {
      vif dc1
    }
  }
}

protocols {
  mld {
    disable: false
    interface dc0 {
      vif dc0 {
        disable: false
      }
    }
    traceoptions {
      flag all {
        disable: false
      }
    }
  }
}
```

In the example configuration above, IGMP is enabled on two vifs on two different interfaces (`dc0/dc0` and `dc1/dc1`). In addition, MLD is enabled on interface/vif `dc0/dc0`, and all MLD tracing functionality is enabled for diagnostic purposes.

10.4 Monitoring IGMP

The `show igmp group` command can be used to display information about IGMP group membership:

```
Xorp> show igmp group
Interface    Group           Source          LastReported Timeout
dc0          224.0.0.2       0.0.0.0         10.4.0.1         161
dc0          224.0.0.13      0.0.0.0         10.4.0.1         159
dc0          224.0.1.20      0.0.0.0         10.4.0.2         197
dc2          224.0.0.2       0.0.0.0         10.3.0.2         155
dc2          224.0.0.13      0.0.0.0         10.3.0.1         157
```

In the above example, `Source` refers to the multicast source address in the case of source-specific IGMP join entries, or it is set to `0.0.0.0` in case of any-source IGMP join entries. The `LastReported` field contains the address of the most recent receiver that responded to an IGMP Join message. The `Timeout` field shows the number of seconds until it is next time to query for host members (*i.e.,* to send an IGMP Query message for this particular entry).

The `show igmp interface` command can be used to display information about IGMP interfaces:

```
Xorp> show igmp interface
Interface    State    Querier         Timeout Version Groups
dc0          UP       10.4.0.1           None       2      3
dc2          UP       10.3.0.1            136       2      2
register_vif DISABLED 0.0.0.0            None       2      0
```

The information indicates whether IGMP is enabled on the interface and the IP address of the IGMP querier. If this router is the querier, then the time until the next query message is shown. Finally the number of multicast groups with receivers on this subnet is shown.

Note that in the above example it is normal for the interface named `register_vif` to be `DISABLED`. This interface has special purpose and is used only by PIM-SM.

The `show igmp interface address` command can be used to display information about addresses of IGMP interfaces:

```
Xorp> show igmp interface address
Interface    PrimaryAddr     SecondaryAddr
dc0          10.4.0.1
dc2          10.3.0.2
register_vif 10.4.0.1
```

As shown above, the `PrimaryAddr` per interface is the address used to originate IGMP messages, and all other alias addresses on that interface are listed as `SecondaryAddr`, with one address per line.

The equivalent commands for MLD are:

- `show mld group`
- `show mld interface`
- `show mld interface address`

Chapter 11

PIM Sparse-Mode

11.1 Terminology and Concepts

PIM stands for *Protocol Independent Multicast*, and denotes a class of multicast routing protocols. The term *protocol independent* comes from the fact that PIM does not have its own topology discovery protocol, but instead relies on routing information supplied by protocols such as RIP and BGP. What PIM does do is to build multicast trees from senders to receivers based on paths determined by this external topology information.

There are two PIM protocols:

- PIM Sparse-Mode (PIM-SM) is the most commonly used multicast routing protocol, and explicitly builds distribution trees from the receivers back towards senders.
- PIM Dense-Mode (PIM-DM) is less commonly used, and builds trees by flooding multicast traffic domain-wide, and then pruning off branches from the tree where there are no receivers.

At the present time, XORP only implements PIM Sparse Mode.

11.1.1 PIM-SM Protocol Overview

The following description is adapted from the PIM-SM specification.

PIM-SM relies on an underlying topology-gathering protocol to populate a routing table with routes. This routing table is called the *MRIB* or *Multicast Routing Information Base*. The routes in this table may be taken directly from the unicast routing table, or it may be different and provided by a separate routing protocol such as Multi-protocol BGP.

Regardless of how it is created, the primary role of the MRIB in the PIM-SM protocol is to provide the next-hop router along a multicast-capable path to each destination subnet. The MRIB is used to determine the next-hop neighbor to which any PIM Join/Prune message is sent. Data flows along the reverse path of the Join messages. Thus, in contrast to the unicast RIB which specifies the next-hop that a data packet would take to get *to* some subnet, the MRIB gives reverse-path information, and indicates the path that a multicast data packet would take *from* its origin subnet to the router that has the MRIB.

Like all multicast routing protocols that implement the ASM service model, PIM-SM must be able to route data packets from sources to receivers without either the sources or receivers knowing a-priori of the existence of the others. This is essentially done in three phases, although as senders and receivers may come and go at any time, all three phases may be occur simultaneously.

Phase One: RP Tree

In phase one, a multicast receiver expresses its interest in receiving traffic destined for a multicast group. Typically it does this using IGMP or MLD. One of the receiver's local PIM routers is elected as the Designated Router (DR) for that subnet. On receiving the receiver's expression of interest, the DR then sends a PIM Join message towards the Rendezvous Point (RP) for that multicast group. The RP is a PIM-SM router that has been configured to serve a bootstrapping role for certain multicast groups. This Join message is known as a (*,G) Join because it joins group G for all sources to that group. The (*,G) Join travels hop-by-hop towards the RP for the group, and in each router it passes through, multicast tree state for group G is instantiated. Eventually the (*,G) Join either reaches the RP, or reaches a router that already has (*,G) Join state for that group. When many receivers join the group, their Join messages converge on the RP, and form a distribution tree for group G that is rooted at the RP. This is known as the RP Tree (RPT), and is also known as the shared tree because it is shared by all sources sending to that group. Join messages are resent periodically so long as the receiver remains in the group. When all receivers on a leaf-network leave the group, the DR will send a PIM (*,G) Prune message towards the RP for that multicast group. However if the Prune message is not sent for any reason, the state will eventually time out.

A multicast data sender just starts sending data destined for a multicast group. The sender's local router (DR) takes those data packets, unicast-encapsulates them, and sends them directly to the RP. The RP receives these encapsulated data packets, decapsulates 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 known as PIM Register packets.

At the end of phase one, multicast traffic is flowing encapsulated to the RP, and then natively over the RP tree to the multicast receivers.

Phase Two: Register-Stop

Register-encapsulation of data packets is inefficient for two reasons:

- Encapsulation and decapsulation may be relatively expensive operations for a router to perform, depending on whether or not the router has appropriate hardware for these tasks.
- Traveling all the way to the RP, and then back down the shared tree may entail the packets traveling a relatively long distance to reach receivers that are close to the sender. For some applications, this increased latency is undesirable.

Although Register-encapsulation may continue indefinitely, for the reasons above, the RP will normally choose to switch to native forwarding. To do this, when the RP receives a register-encapsulated data packet from source S on group G, it will normally initiate an (S,G) source-specific Join towards S. This Join message travels hop-by-hop towards S, instantiating (S,G) multicast tree state in the routers along the path. (S,G) multicast tree state is used only to forward packets for group G if those packets come from source S. Eventually the Join message reaches S's subnet or a router that already has (S,G) multicast tree state, and then packets from S start to flow following the (S,G) tree state towards the RP. These data packets may also reach routers with (*,G) state along the path towards the RP - if so, they can short-cut onto the RP tree at this point.

While the RP is in the process of joining the source-specific tree for S, the data packets will continue being encapsulated to the RP. When packets from S also start to arrive natively at the the RP, the RP will be receiving two copies of each of these packets. At this point, the RP starts to discard the encapsulated copy of these packets, and it sends a *Register-Stop* message back to S's DR to prevent the DR unnecessarily encapsulating the packets.

At the end of phase 2, traffic will be flowing natively from S along a source-specific tree to the RP, and from there along the shared tree to the receivers. Where the two trees intersect, traffic may transfer from the source-specific tree to the RP tree, and so avoid taking a long detour via the RP.

It should be noted that a sender may start sending before or after a receiver joins the group, and thus phase two may happen before the shared tree to the receiver is built.

Phase 3: Shortest-Path Tree

Although having the RP join back towards the source removes the encapsulation overhead, it does not completely optimize the forwarding paths. For many receivers the route via the RP may involve a significant detour when compared with the shortest path from the source to the receiver.

To obtain lower latencies, a router on the receiver's LAN, typically the DR, may optionally initiate a transfer from the shared tree to a source-specific shortest-path tree (SPT). To do this, it issues an (S,G) Join towards S. This instantiates state in the routers along the path to S. Eventually this join either reaches S's subnet, or reaches a router that already has (S,G) state. When this happens, data packets from S start to flow following the (S,G) state until they reach the receiver.

At this point the receiver (or a router upstream of the receiver) will be receiving two copies of the data - one from the SPT and one from the RPT. When the first traffic starts to arrive from the SPT, the DR or upstream router starts to drop the packets for G from S that arrive via the RP tree. In addition, it sends an (S,G) Prune message towards the RP. This is known as an (S,G,rpt) Prune. The Prune message travels hop-by-hop, instantiating state along the path towards the RP indicating that traffic from S for G should NOT be forwarded in this direction. The prune is propagated until it reaches the RP or a router that still needs the traffic from S for other receivers.

By now, the receiver will be receiving traffic from S along the shortest-path tree between the receiver and S. In addition, the RP is receiving the traffic from S, but this traffic is no longer reaching the receiver along the RP tree. As far as the receiver is concerned, this is the final distribution tree.

Multi-access Transit LANs

The overview so far has concerned itself with point-to-point links. However, using multi-access LANs such as Ethernet for transit is not uncommon. This can cause complications for three reasons:

- Two or more routers on the LAN may issue (*,G) Joins to different upstream routers on the LAN because they have inconsistent MRIB entries regarding how to reach the RP. Both paths on the RP tree will be set up, causing two copies of all the shared tree traffic to appear on the LAN.

- Two or more routers on the LAN may issue (S,G) Joins to different upstream routers on the LAN because they have inconsistent MRIB entries regarding how to reach source S. Both paths on the source-specific tree will be set up, causing two copies of all the traffic from S to appear on the LAN.

- A router on the LAN may issue a (*,G) Join to one upstream router on the LAN, and another router on the LAN may issue an (S,G) Join to a different upstream router on the same LAN. Traffic from S may reach the LAN over both the RPT and the SPT. If the receiver behind the downstream (*,G) router doesn't issue an (S,G,rpt) prune, then this condition would persist.

All of these problems are caused by there being more than one upstream router with join state for the group or source-group pair. PIM-SM does not prevent such duplicate joins from occurring - instead when duplicate data packets appear on the LAN from different routers, these routers notice this, and then elect a single forwarder. This election is performed using PIM *Assert* messages, which resolve the problem in favor of the upstream router which has (S,G) state, or if neither or both router has (S,G) state, then in favor of the router with the best metric to the RP for RP trees, or the best metric to the source to source-specific trees.

These Assert messages are also received by the downstream routers on the LAN, and these cause subsequent Join messages to be sent to the upstream router that won the Assert.

RP Discovery

PIM-SM routers need to know the address of the RP for each group for which they have (*,G) state. This address is obtained either through a bootstrap mechanism or through static configuration.

One dynamic way to do this is to use the *Bootstrap Router* (BSR) mechanism. One router in each PIM-SM domain is elected the Bootstrap Router through a simple election process. All the routers in the domain that are configured to be candidates to be RPs periodically unicast their candidacy to the BSR. From the candidates, the BSR picks an RP-set, and periodically announces this set in a Bootstrap message. Bootstrap messages are flooded hop-by-hop throughout the domain until all routers in the domain know the RP-Set.

To map a group to an RP, a router hashes the group address into the RP-set using an order-preserving hash function (one that minimizes changes if the RP-Set changes). The resulting RP is the one that it uses as the RP for that group.

11.2 Standards

XORP is compliant with the following PIM-SM specification:

draft-ietf-pim-sm-v2-new-10. Protocol Independent Multicast - Sparse Mode (PIM-SM): Protocol Specification (Revised).

draft-ietf-pim-sm-bsr-02. Bootstrap Router (BSR) Mechanism for PIM Sparse Mode.

11.3 Configuring PIM-SM

11.3.1 Configuring Multicast Routing on UNIX Systems

If XORP is to be run on a UNIX-based system, the following steps must be taken to enable the system for PIM-SM multicast routing before starting XORP:

- Make sure that the underlying system supports multicast routing and has PIM-SM kernel support. Unfortunately, there is no trivial guideline how to check this, but the following OS-specific information can be useful:
  - `DragonFlyBSD`: DragonFlyBSD-1.0 and later.
  - `FreeBSD`: IPv4 (FreeBSD-4.9 and later, FreeBSD-5.2 and later), IPv6 (FreeBSD-4.x and later).
  - `Linux`: IPv4 (Linux-2.2.11 and later, Linux-2.3.6 and later), IPv6 (only with the IPv6 USAGI toolkit after 2005/02/14: http://www.linux-ipv6.org/).
  - `MacOS X`: No multicast routing support (as of MacOS X 10.3.x).
  - `NetBSD`: IPv4 (any release after NetBSD-2.0), IPv6 (NetBSD-1.5 and later).
  - `OpenBSD`: IPv4 (OpenBSD-3.7 and later), IPv6 (OpenBSD-2.7 and later).
- If necessary, configure the kernel to enable multicast routing and PIM-SM:
  - `DragonFlyBSD`:
    IPv4: enable the following options in the kernel:

    ```
    options         MROUTING                    # Multicast routing
    options         PIM                         # PIM multicast routing
    ```

    IPv6: no kernel options are required.
  - `FreeBSD`:
    IPv4: enable the following options in the kernel:

    ```
    options         MROUTING                    # Multicast routing
    options         PIM                         # PIM multicast routing
    ```

    IPv6: no kernel options are required.
  - `Linux`:
    IPv4: enable the following options in the kernel:

    ```
    CONFIG_IP_MULTICAST=y
    CONFIG_IP_MROUTE=y
    CONFIG_IP_PIMSM_V2=y
    ```

    IPv6: Enable the following options in the kernel:

    ```
    CONFIG_IPV6_MROUTE=y
    CONFIG_IPV6_PIMSM_V2=y
    ```

- NetBSD:
  IPv4: enable the following options in the kernel:

  ```
  options         MROUTING        # IP multicast routing
  options         PIM             # Protocol Independent Multicast
  ```

  IPv6: no kernel options are required.
- OpenBSD:
  IPv4: enable the following options in the kernel:

  ```
  option          MROUTING        # Multicast router
  option          PIM             # Protocol Independent Multicast
  ```

  IPv6: no kernel options are required.

• Apply additional system configuration (if necessary):

- DragonFlyBSD:
  IPv4: Enable IPv4 unicast forwarding:

  ```
  sysctl net.inet.ip.forwarding=1
  ```

  IPv6: Enable IPv6 unicast forwarding:

  ```
  sysctl net.inet6.ip6.forwarding=1
  ```
- FreeBSD:
  IPv4: Enable IPv4 unicast forwarding:

  ```
  sysctl net.inet.ip.forwarding=1
  ```

  IPv6: Enable IPv6 unicast forwarding:

  ```
  sysctl net.inet6.ip6.forwarding=1
  ```
- Linux:
  IPv4: Enable IPv4 unicast forwarding:

  ```
  echo 1 > /proc/sys/net/ipv4/ip_forward
  ```

  If the unicast Reverse Path Forwarding information is different from the multicast Reverse Path Forwarding information, the Reverse Path Filtering should be disabled:

  ```
  echo 0 > /proc/sys/net/ipv4/conf/all/rp_filter
  ```

  OR

  ```
  echo 0 > /proc/sys/net/ipv4/conf/eth0/rp_filter
  echo 0 > /proc/sys/net/ipv4/conf/eth1/rp_filter
  ...
  ```

  IPv6: unknown
- NetBSD: none.

– OpenBSD: Add the following lines to /etc/rc.conf.local and reboot:

```
# Enable multicast routing (see netstart(8) for details).
multicast_host=NO
multicast_router=YES
```

11.3.2 Configuration Syntax

```
protocols {
  pimsm4 {
    targetname: text
    disable: bool
    interface text {
      vif text {
        disable: bool
        dr-priority: uint
        alternative-subnet IPv4/int(0..32)
      }
    }
    interface register_vif {
      vif register_vif {
        disable: bool
      }
    }

    static-rps {
      rp IPv4 {
        group-prefix IPv4Mcast/int(4..32) {
          rp-priority: uint(0..255)
          hash-mask-len: uint(4..32)
        }
      }
    }

    bootstrap {
      disable: bool
      cand-bsr {
        scope-zone IPv4Mcast/int(4..32) {
          is-scope-zone: bool
          cand-bsr-by-vif-name: text
          bsr-priority: uint(0..255)
          hash-mask-len: uint(4..32)
        }
      }

      cand-rp {
        group-prefix IPv4Mcast/int(4..32) {
          is-scope-zone: bool
          cand-rp-by-vif-name: text
          rp-priority: uint(0..255)
          rp-holdtime: uint(0..65535)
        }
      }
    }

    switch-to-spt-threshold {
      disable: bool
      interval-sec: uint(3..2147483647)
      bytes: uint
    }
```

continued overleaf....

```
    traceoptions {
      flag all {
        disable: bool
      }
    }
  }
}

protocols {
  pimsm6 {
    disable: bool
    interface text {
      vif text {
        disable: bool
        dr-priority: uint
        alternative-subnet IPv6/int(0..128)
      }
    }
    interface register_vif {
      vif register_vif {
        disable: bool
      }
    }

    static-rps {
      rp IPv6 {
        group-prefix IPv6Mcast/int(8..128) {
          rp-priority: uint(0..255)
          hash-mask-len: uint(8..128)
        }
      }
    }

    bootstrap {
      disable: bool
      cand-bsr {
        scope-zone IPv6Mcast/int(8..128) {
          is-scope-zone: bool
          cand-bsr-by-vif-name: text
          bsr-priority: uint(0..255)
          hash-mask-len: uint(8..128)
        }
      }

      cand-rp {
        group-prefix IPv6Mcast/int(8..128) {
          is-scope-zone: bool
          cand-rp-by-vif-name: text
          rp-priority: uint(0..255)
          rp-holdtime: uint(0..65535)
        }
      }
    }

    switch-to-spt-threshold {
      disable: bool
      interval-sec: uint(3..2147483647)
      bytes: uint
    }

    traceoptions {
      flag all {
        disable: bool
      }
    }
  }
}
```

`protocols`: this delimits the configuration for all routing protocols in the XORP router configuration. It is mandatory that PIM-SM configuration is under the `protocols` node in the configuration.

`pimsm4`: this delimits the PIM-SM configuration part of the XORP router configuration related to IPv4 multicast.

`targetname`: this is the name for this instance of PIM-SM for IPv4. It defaults to "`PIMSM_4`", and it is not recommended that this default is overridden under normal usage scenarios.

`disable`: this takes the value `true` or `false`, and indicates whether PIM-SM IPv4 multicast routing is currently disabled [1]. This allows multicast to be taken down temporarily without removing the configuration.

`interface`: this directive specifies that this `interface` is to be used for PIM-SM IPv4 multicast routing. The parameter value must be the name of an interface that has been configured in the `interfaces` section of the router configuration.

`vif`: this directive specifies that this `vif` on the specified `interface` is to be used for PIM-SM IPv4 multicast routing. The parameter value must be the name of a vif that has been configured in the `interfaces` section of the router configuration.

A special logical interface called `register_vif` with a special vif called `register_vif` must be configured if a PIM-SM router is to be able to send Register messages to the RP. In general this should *always* be configured if the router is to support the ASM multicast service model.

Each `vif` can take the following optional parameters:

`disable`: this takes the value `true` or `false`, and indicates whether PIM-SM IPv4 multicast routing is currently disabled on this interface/vif [2].

`dr-priority`: this directive takes a non-negative integer as its parameter giving this router's Designated Router (DR) priority for this interface/vif. The default is 1. The PIM router on this subnet with the highest value of DR priority will become the DR for the subnet.

`alternative-subnet`: this directive is used to associate additional IP subnets with a network interface. The parameter value is an IPv4 subnet address in the *address/prefix-length* format.

One use of this directive is to make incoming traffic with a non-local source address appear as it is coming from a local subnet. Typically, this is needed as a work-around solution when uni-directional interfaces such as satellite links are used for receiving traffic. The `alternative-subnet` directive should be used with extreme care, because it is possible to create forwarding loops.

`static-rps`: this delimits the part of the PIM-SM configuration used to manually configure PIM RP router information. A PIM-SM router must either have some RPs configured as static RPs, or it must run the PIM-SM bootstrap mechanism (see the `bootstrap` directive).

Under the `static-rps` part of the configuration, one or more RPs can be configured. It is important that all routers in a PIM domain make the same choice of RP for the same multicast group, so generally they should be configured with the same RP information.

`rp`: this specifies the IPv4 address of a router to be a static RP.

For each RP, the following parameters can be configured:

[1] Note that prior to XORP Release-1.1, the `enable` flag was used instead of `disable`.
[2] Note that prior to XORP Release-1.1, the `enable` flag was used instead of `disable`.

`group-prefix`: this specifies the range of multicast addresses for which the specified router is willing to be the RP. The value is in the form of an IP address and prefix-length in the *address/prefix-length* format.

`rp-priority`: this specifies the priority of the specified RP router. It takes the form of a non-negative integer in the interval [0, 255].
If multiple RP routers are known for a particular multicast group, then the one with the most specific `group-prefix` will be used. If more than one router has the same most specific `group-prefix`, then the one with the highest `rp-priority` is used. See also `hash-mask-len`.
The default value is 192.

`hash-mask-len`: If multiple routers have the most specific `group-prefix` and the same highest `rp-priority`, then to balance load, a hash function is used to choose the RP. However, it is usually desirable for closely associated multicast groups to use the same RP. Thus the hash function is only applied to the first n bits of the group IP address, ensuring that if two groups have the same first n bits, they will hash to the same RP address. The `hash-mask-len` parameter specifies the value of n. For IPv4 it must be in the interval [4, 32], and defaults to 30 bits. Typically its value shouldn't be changed. If it is modified then all PIM-SM routers must be configured with the same value.

`bootstrap`: this delimits the part of the PIM-SM configuration used to configure the automatic bootstrap of PIM RP router information using the PIM *BootStrap Router* mechanism. A PIM-SM router must either run the PIM-SM bootstrap mechanism, or have some RPs configured as static RPs (see the `static-rps` directive).

Under the `bootstrap` directive, the following additional information can be configured.

`disable`: this takes the value `true` or `false`, and determines whether or not the router will run the `bootstrap` mechanism [3]. The default is `false`.

`cand-bsr`: this directive specifies that this router is to be a candidate to be the BootStrap Router (BSR) for this PIM-SM domain. It will become the BSR only if it wins the BSR election process.

One or more `scope-zone`s must be specified for a candidate BSR router:

`scope-zone`: this directive specifies one multicast group prefix for which this router is willing to be BSR.
For each scope zone, the following information can be specified:

`is-scope-zone`: this directive takes the value `true` or `false`. When the value is `true`, this indicates that this multicast group prefix defines a multicast scope zone. When the value is `false`, this indicates that the group prefix in the `scope-zone` directive merely represents a range of multicast groups for which this router is willing to be BSR. The default is `false`.

`cand-bsr-by-vif-name`: this specifies the name of the `vif` whose IP address will be used in the PIM bootstrap messages. It is a mandatory parameter.

`bsr-priority`: this specifies the BSR priority for this router. It takes a positive integer value in the interval [0, 255], which is used in the PIM-SM BSR election process. For each `scope-zone`, the candidate bootstrap router with the highest BSR priority will be chosen to be BSR. Its default value is 1.

[3] Note that prior to XORP Release-1.1, the `enable` flag was used instead of `disable`.

hash-mask-len: The BSR mechanism announces a list of candidate RPs (C-RPs) for each scope zone to the other routers in the scope zone. To balance load, those routers then use a hash function to choose the RP for each multicast group from amongst the C-RPs. However, it is usually desirable for closely associated multicast groups to use the same RP. Thus the hash function is only applied to the first n bits of the group IP address, ensuring that if two groups have the same first n bits, they will hash to the same RP address. Should this router become the BSR for this scope-zone, the hash-mask-len parameter gives the value of n that this router will inform other routers they must use. For IPv4 it must be in the interval [4, 32], and defaults to 30 bits. Typically its value shouldn't be changed. If it is modified then all PIM-SM routers must be configured with the same value.

cand-rp: this directive specifies that this router is to be a candidate to be an RP for this PIM-SM domain. It will become an RP only if the BSR chooses it to be.

One or more group-prefixes must be specified for this router to function as an RP:

group-prefix: this specifies the range of multicast addresses for which the specified router is willing to be the RP. The value is in the form of an IP address and prefix length in the *address/prefix-length* format.

For each group-prefix, the following parameters can be specified:

is-scope-zone: this directive takes the value true or false. When the value is true, this indicates that this multicast group prefix defines a multicast scope zone. When the value is false, this indicates that the group prefix in the scope-zone directive merely represents a range of multicast groups for which this router is willing to be RP. The default is false.

cand-rp-by-vif-name: this specifies the name of the vif whose IP address will be used as the RP address if this router becomes an RP. It is a mandatory parameter.

rp-priority: this specifies the RP priority of this router for this group-prefix. It takes the form of a non-negative integer in the interval [0, 255].
If multiple RP routers are known for a particular multicast group, then the one with the most specific group-prefix will be used. If more than one router has the same most specific group-prefix, then the one with the highest rp-priority is used. See also hash-mask-len.
The default value for rp-priority is 1.

rp-holdtime: this specifies the holdtime that this router will advertise when talking to the BSR. If the BSR has not heard a Candidate RP Advertisement from this router for rp-holdtime seconds, then the BSR will conclude it is dead, and will remove it from the set of possible RPs. It takes the form of a non-negative integer in the interval [0, 65535] and its default value is 150 seconds.

switch-to-spt-threshold: this directive permits the specification of a bitrate threshold at a last-hop router or RP for switching from the RP Tree to the Shortest-Path Tree. The following parameters can be specified:

disable: this takes the value true or false, and determines whether bitrate-based switching to the shortest path tree is disabled [4]. The default is false.

[4] Note that prior to XORP Release-1.1, the enable flag was used instead of disable.

`interval-sec`: this specifies the measurement interval in seconds for measuring the bitrate of traffic from a multicast sender. The measurement interval should normally not be set too small - values greater than ten seconds are recommended. It takes the form of a non-negative integer in the interval [3, 2147483647] and its default value is 100 seconds.

`bytes`: this specifies the maximum number of bytes from a multicast sender that can be received in `interval-sec` seconds. If this threshold is exceeded, the router will attempt to switch to the shortest-path tree from that multicast sender. If the shortest-path switch should happen right after the first packet is forwarded, then `bytes` should be set to 0.

`traceoptions`: this directive delimits the configuration of debugging and tracing options for PIM-SM.

`flag`: this directive is used to specify which tracing options are enabled. Possible parameters are:

`all`: this directive specifies that all tracing options should be enabled. Possible parameters are:

`disable`: this takes the value `true` or `false`, and disables or enables tracing [5]. The default is `false`.

The configuration parameters for PIM-SM for IPv6 are the same as those for IPv4, except for the following:

- The `pimsm6` directive for IPv6 is used in place of the `pimsm4` directive for IPv4.
- The default value of `targetname` is "`PIMSM_6`" instead of "`PIMSM_4`"
- All IP addresses used in configuration are IPv6 addresses instead of IPv4 addresses.
- The `hash-mask-len` value must be in the interval [8, 128], and defaults to 126.

[5] Note that prior to XORP Release-1.1, the `enable` flag was used instead of `disable`.

11.3.3 Example Configurations

```
protocols {
  pimsm4 {
    disable: false
    interface dc0 {
      vif dc0 {
        disable: false
        /* dr-priority: 1 */
        /* alternative-subnet 10.40.0.0/16 */
      }
    }
    interface register_vif {
      vif register_vif {
        /* Note: this vif should be always enabled */
        disable: false
      }
    }

    static-rps {
      rp 10.60.0.1 {
        group-prefix 224.0.0.0/4 {
          /* rp-priority: 192 */
          /* hash-mask-len: 30 */
        }
      }
    }

    bootstrap {
      disable: false
      cand-bsr {
        scope-zone 224.0.0.0/4 {
          /* is-scope-zone: false */
          cand-bsr-by-vif-name: "dc0"
          /* bsr-priority: 1 */
          /* hash-mask-len: 30 */
        }
      }

      cand-rp {
        group-prefix 224.0.0.0/4 {
          /* is-scope-zone: false */
          cand-rp-by-vif-name: "dc0"
          /* rp-priority: 192 */
          /* rp-holdtime: 150 */
        }
      }
    }

    switch-to-spt-threshold {
      /* approx. 1K bytes/s (10Kbps) threshold */
      disable: false
      interval-sec: 100
      bytes: 102400
    }
```

continued overleaf....

```
    traceoptions {
      flag all {
        disable: false
      }
    }
  }
}

protocols {
  pimsm6 {
    disable: false
    interface dc0 {
      vif dc0 {
        disable: false
        /* dr-priority: 1 */
        /* alternative-subnet 40:40:40:40::/64 */
      }
    }
    interface register_vif {
      vif register_vif {
        /* Note: this vif should be always enabled */
        disable: false
      }
    }

    static-rps {
      rp 50:50:50:50:50:50:50:50 {
        group-prefix ff00::/8 {
          /* rp-priority: 192 */
          /* hash-mask-len: 126 */
        }
      }
    }

    bootstrap {
      disable: false
      cand-bsr {
        scope-zone ff00::/8 {
          /* is-scope-zone: false */
          cand-bsr-by-vif-name: "dc0"
          /* bsr-priority: 1 */
          /* hash-mask-len: 30 */
        }
      }

      cand-rp {
        group-prefix ff00::/8 {
          /* is-scope-zone: false */
          cand-rp-by-vif-name: "dc0"
          /* rp-priority: 192 */
          /* rp-holdtime: 150 */
        }
      }
    }

    switch-to-spt-threshold {
      /* approx. 1K bytes/s (10Kbps) threshold */
      disable: false
      interval-sec: 100
      bytes: 102400
    }

    traceoptions {
      flag all {
        disable: false
      }
    }
  }
}
```

11.4 Monitoring PIM-SM

All operational commands for monitoring PIM-SM for IPv4 begin with `show pim`. This section describes those commands in details. All operational commands for monitoring PIM-SM for IPv6 are similar except that they begin with `show pim6`.

11.4.1 Monitoring PIM-SM Bootstrap Information

The `show pim bootstrap` command can be used to display information about PIM bootstrap routers:

```
Xorp> show pim bootstrap
Active zones:
BSR            Pri LocalAddress      Pri State       Timeout SZTimeout
10.4.0.1         1 10.2.0.2            1 Candidate        75        -1
Expiring zones:
BSR            Pri LocalAddress      Pri State       Timeout SZTimeout
Configured zones:
BSR            Pri LocalAddress      Pri State       Timeout SZTimeout
10.2.0.2         1 10.2.0.2            1 Init             -1        -1
```

The bootstrap information is separated in three sections:

- `Active zones:` This section contains the bootstrap zones that are currently in use.
- `Expiring zones:` If new bootstrap information is received and it replaces the old bootstrap information, the old information is deleted. However, if some of the old bootstrap information was not replaced, that information is moved to the `Expiring zones` section until it times out.
- `Configured zones:` This section contains the bootstrap zones that are configured on the router.

The fields for each entry (in order of appearance) are:

- `BSR:` The address of the Bootstrap router for the zone.
- `Pri:` The priority of the Bootstrap router.
- `LocalAddress:` The local Candidate-BSR address for the zone (if the router is configured as a Candidate-BSR).
- `Pri:` The local Candidate-BSR priority for the zone (if the router is configured as a Candidate-BSR).
- `State:` The state of the per-scope-zone state machine. In the above example, the router is configured as a Candidate-BSR, but it is not the elected BSR, hence its state is `Candidate`.
- `Timeout:` The number of seconds until the BSR times-out. If it is -1, it will never timeout.
- `SZTimeout:` The number of seconds until the scoped zone times-out. If it is -1, it will never timeout.

The `show pim bootstrap rps` command can be used to display information about Candidate RP information received by the Bootstrap mechanism:

```
Xorp> show pim bootstrap rps
Active RPs:
RP              Pri Timeout GroupPrefix BSR         CandRpAdvTimeout
10.4.0.1        192     148 224.0.0.0/4 10.4.0.1                  -1
10.2.0.2        192     148 224.0.0.0/4 10.4.0.1                  -1
Expiring RPs:
RP              Pri Timeout GroupPrefix BSR         CandRpAdvTimeout
Configured RPs:
RP              Pri Timeout GroupPrefix BSR         CandRpAdvTimeout
10.2.0.2        192      -1 224.0.0.0/4 10.2.0.2                  58
```

The Candidate RPs information is separated in three sections:

- `Active RPs:` This section contains the Candidate RPs that are currently in use.
- `Expiring RPs:` If new bootstrap information is received and it replaces the old bootstrap information, the old information is deleted. However, if some of the old bootstrap information was not replaced, the Candidate RPs contained in that information are moved to the `Expiring RPs` section until they time-out.
- `Configured RPs:` This section contains the Candidate RP information that is configured on the router.

The fields for each entry (in order of appearance) are:

- `RP:` The address of the Candidate RP for the entry.
- `Pri:` The priority of the Candidate RP.
- `Timeout:` The number of seconds until the Candidate RP times-out. If it is -1, it will never timeout.
- `GroupPrefix:` The multicast group prefix address the Candidate RP is advertising.
- `BSR:` The address of the BSR that advertised this Candidate RP.
- `CandRpAdvTimeout:` The number of seconds until the Candidate RP is advertised to the BSR. This applies only for the Candidate-RPs configured in this router. If it is -1, the Candidate RP is not advertised to the BSR.

11.4.2 Monitoring PIM-SM Interface Information

The `show pim interface` command can be used to display information about PIM network interfaces:

```
Xorp> show pim interface
Interface    State    Mode   V PIMstate Priority DRaddr         Neighbors
dc1          UP       Sparse 2 NotDR           1 10.3.0.2               1
dc2          UP       Sparse 2 DR              1 10.2.0.2               0
register_vif UP       Sparse 2 DR              1 10.3.0.1               0
```

The fields for each entry (in order of appearance) are:

- `Interface`: The name of the interface.
- `State`: The state of the interface. E.g. `UP`, `DOWN`, `DISABLED`, etc.
- `Mode`: The PIM mode of the interface. E.g. `Sparse` means PIM-SM.
- `V`: The protocol version.
- `PIMstate`: The protocol state on that interface. E.g., `DR` means the router is the Designated Router on that interface.
- `Priority`: The configured Designated Router priority on that interface.
- `DRaddr`: The address of the elected Designated Router on the subnet connected to that interface.
- `Neighbors`: The number of PIM neighbor routers on that interface.

The `show pim interface address` command can be used to display address information about PIM network interfaces:

```
Xorp> show pim interface address
Interface    PrimaryAddr     DomainWideAddr  SecondaryAddr
dc1          10.3.0.1        10.3.0.1
dc2          10.2.0.2        10.2.0.2
register_vif 10.3.0.1        10.3.0.1
```

The fields for each entry (in order of appearance) are:

- `Interface`: The name of the interface.
- `PrimaryAddr`: The primary address on the interface.
- `DomainWideAddr`: The domain-wide address on the interface.
- `SecondaryAddr`: The first secondary address on the interface (if any). If there is more than one secondary address on the interface, they are printed one per new line (in the same column).

11.4.3 Monitoring PIM-SM Multicast Routing State Information

The `show pim join` command can be used to display information about PIM multicast routing state:

```
Xorp> show pim join
Group           Source          RP            Flags
224.0.1.20      0.0.0.0         10.2.0.2      WC
    Upstream interface (RP):   register_vif
    Upstream MRIB next hop (RP): UNKNOWN
    Upstream RPF'(*,G):        UNKNOWN
    Upstream state:            Joined
    Join timer:                21
    Local receiver include WC: .O.
    Joins RP:                  ...
    Joins WC:                  ...
    Join state:                ...
    Prune state:               ...
    Prune pending state:       ...
    I am assert winner state:  ...
    I am assert loser state:   ...
    Assert winner WC:          ...
    Assert lost WC:            ...
    Assert tracking WC:        .OO
    Could assert WC:           .O.
    I am DR:                   .OO
    Immediate olist RP:        ...
    Immediate olist WC:        .O.
    Inherited olist SG:        .O.
    Inherited olist SG_RPT:    .O.
    PIM include WC:            .O.
```

The fields for each entry (in order of appearance) are:

- `Group`: The group address.
- `Source`: The source address.
- `RP`: The address of the RP for this entry.
- `Flags`: The set of flags for this entry. For example:
  - `RP`: (*,*,RP) routing entry.
  - `WC`: (*,G) routing entry.
  - `SG`: (S,G) routing entry.
  - `SG_RPT`: (S,G,rpt) routing entry.
  - `SPT`: The routing entry has the Shortest-Path Tree flag set.
  - `DirectlyConnectedS`: The routing entry is for a directly-connected source.

The remaining lines per entry display various additional information for that entry. Some of the information below contains a set of network interfaces: there is either "." or "O" per interface (starting with the first interface according to the `show pim interface` command), and if an interface is included, it is marked with "O".

- `Upstream interface (RP)`: The name of the upstream interface toward the RP.
- `Upstream MRIB next hop (RP)`: The address of the next-hop router (according to the MRIB) toward the RP. In the above example the router itself is the RP, hence there is no next-hop router.

- `Upstream RPF'(*,G)`: The address of the next-hop router (according to PIM) toward the RP. Note that this address may be different, because it may be affected by PIM-specific events such as PIM Assert messages on the upstream interface. In the above example the router itself is the RP, hence there is no next-hop router.
- `Upstream state`: The upstream state of this entry.
- `Join timer`: The number of seconds until the upstream Join timer timeout.
- `Local receiver include WC`: The set of interfaces that have local (*,G) receivers according to the MLD/IGMP module.
- `Joins RP`: The set of interfaces that have received (*,*,RP) Join.
- `Joins WC`: The set of interfaces that have received (*,G) Join.
- `Join state`: The set of interfaces that are in Join state.
- `Prune state`: The set of interfaces that are in Prune state.
- `Prune pending state`: The set of interfaces that are in Prune-Pending state.
- `I am assert winner state`: The set of interfaces that are in Assert Winner state.
- `I am assert loser state`: The set of interfaces that are in Assert Loser state.
- `Assert winner WC`: The set of interfaces for which the corresponding (*,G) entry is in Assert Winner state.
- `Assert lost WC`: The set of interfaces for which the corresponding (*,G) entry has lost the PIM Assert.
- `Assert tracking WC`: The set of interfaces for which the corresponding (*,G) entry desires to track the PIM Asserts.
- `Could assert WC`: The set of interfaces for which the corresponding (*,G) entry could trigger a PIM Assert.
- `I am DR`: The set of interfaces for which this is the Designated Router.
- `Immediate olist RP`: The set of interfaces that are included in the immediate outgoing interfaces for the corresponding (*,*,RP) entry.
- `Immediate olist WC`: The set of interfaces that are included in the immediate outgoing interfaces for the corresponding (*,RP) entry.
- `Inherited olist SG`: The set of interfaces that are included in the outgoing interface list for packets forwarded on (S,G) state taking into account (*,*,RP) state, (*,G) state, asserts, etc.
- `Inherited olist SG_RPT`: The set of interfaces that are included in the outgoing interface list for packets forwarded on (*,*,RP) or (*,G) state taking into account (S,G,rpt) prune state, and asserts, etc.
- `PIM include WC`: The set of interfaces to which traffic might be forwarded because of hosts that are local members on that interface.

The `show pim join all` command can be used to display information about all PIM multicast routing entries including those that may be created internally by the PIM implementation. Typically, those are the (*,*,RP) entries that are created per RP for implementation-specific reasons even though there is no requirement to do so. Currently, this command is used only for debugging purpose.

11.4.4 Monitoring PIM-SM Multicast Routing State Information

The `show pim mfc` command can be used to display information about PIM multicast forwarding entries that are installed in the multicast forwarding engine:

```
Xorp> show pim mfc
Group           Source          RP
224.0.1.20      10.4.0.2        10.2.0.2
    Incoming interface :       register_vif
    Outgoing interfaces:       .O.
```

The fields for each entry (in order of appearance) are:

- `Group:` The group address.
- `Source:` The source address.
- `RP:` The address of the RP for this entry.

The remaining lines per entry display various additional information for that entry. Some of the information below contains a set of network interfaces: there is either "." or "O" per interface (starting with the first interface according to the `show pim interface` command), and if an interface is included, it is marked with "O".

- `Incoming interface:` The name of the incoming interface.
- `Outgoing interfaces:` The set of outgoing interfaces.

11.4.5 Monitoring PIM-SM Multicast Routing Information Base

The `show pim mrib` command can be used to display information about the Multicast Routing Information Base (MRIB) that is used by PIM:

```
Xorp> show pim mrib
DestPrefix         NextHopRouter  VifName VifIndex MetricPref Metric
10.2.0.0/24        10.2.0.2       dc2     1                 0      0
10.3.0.0/24        10.3.0.1       dc1     0                 0      0
10.4.0.0/24        10.3.0.2       dc1     0               254  65535
10.5.0.0/24        10.2.0.4       dc2     1               254  65535
10.6.0.0/24        10.2.0.1       dc2     1               254  65535
```

The fields for each entry (in order of appearance) are:

- `DestPrefix:` The destination prefix address.

- `NextHopRouter`: The address of the next-hop router toward the destination.
- `VifName`: The name of the virtual interface toward the destination.
- `VifIndex`: The virtual interface index of the virtual interface toward the destination.
- `MetricPref`: The metric preference of the entry.
- `Metric`: The routing metric of the entry.

11.4.6 Monitoring PIM-SM Multicast Routing Information Base

The `show pim neighbors` command can be used to display information about the PIM neighbor routers:

```
Xorp> show pim neighbors
Interface    DRpriority NeighborAddr    V Mode   Holdtime Timeout
dc1                   1 10.3.0.2        2 Sparse      105      97
```

The fields for each entry (in order of appearance) are:

- `Interface`: The name of the interface toward the neighbor:
- `DRpriority`: The DR priority of the neighbor.
- `NeighborAddr`: The primary address of the neighbor.
- `V`: The PIM protocol version used by the neighbor.
- `Mode`: The PIM mode of the neighbor. E.g. `Sparse` means PIM-SM.
- `Holdtime`: The PIM Hello holdtime of the neighbor (in seconds).
- `Timeout`: The number of seconds until the neighbor timeout (in case no more PIM Hello messages are received from it).

11.4.7 Monitoring PIM-SM Candidate RP Set Information

The `show pim rps` command can be used to display information about the Candidate RP Set:

```
Xorp> show pim rps
RP              Type      Pri Holdtime Timeout ActiveGroups GroupPrefix
10.4.0.1        bootstrap 192      150     134            0 224.0.0.0/4
10.2.0.2        bootstrap 192      150     134            1 224.0.0.0/4
```

The fields for each entry (in order of appearance) are:

- `RP`: The address of the Candidate RP.
- `Type`: The type of the mechanism that provided the Candidate RP.
- `Pri`: The priority of the Candidate RP.

- `Holdtime:` The holdtime (in number of seconds) of the Candidate RP.
- `Timeout:` The number of seconds until the Candidate RP timeout. If it is -1, the Candidate RP will never timeout.
- `ActiveGroups:` The number of groups that use this Candidate RP.
- `GroupPrefix:` The multicast group prefix address for this Candidate RP.

11.4.8 Monitoring PIM-SM Scope Zone Information

The `show pim scope` command can be used to display information about the PIM scope zones:

```
Xorp> show pim scope
GroupPrefix                                  Interface
225.1.2.0/24                                 dc1
```

The fields for each entry (in order of appearance) are:

- `GroupPrefix:` The multicast group prefix address of the scoped zone.
- `Interface:` The name of the interface that is the boundary of the scoped zone.

Note that currently (April 2005), configuring multicast scoped zones is not supported. This feature should be added in the future.

Chapter 12

Multicast Topology Discovery

12.1 Terminology and Concepts

Multicast routing protocols such as PIM-SM (Protocol Independent Multicast Sparse-Mode) and PIM-DM (Protocol Independent Multicast Dense-Mode) build the multicast delivery tree by using the RPF (Reverse-Path Forwarding) information toward the root of the tree. The root could be the so-called Rendezvous Point (RP) (in case of PIM-SM) or the source itself (in case of PIM-SM or PIM-DM).

The RPF information in each router is per multicast distribution tree and is basically the next-hop neighbor router information toward the root of the tree. In other words, the RPF router is the next-hop router toward the root. In case of PIM-SM, the RPF neighbor is typically the router that a Join message is sent to.

Obviously, all multicast routers must have consistent RPF state, otherwise a Join message may never reach the root of the tree. Typically, the unicast path forwarding information is used to create the RPF information, because under normal circumstances the unicast routing provides the necessary information to all routers.

Note that he unicast-based RPF creates multicast distribution trees where each branch of the tree follows the unicast path from each leaf of the tree toward the root. Usually this is the desired behavior, but occasionally someone may want the unicast and the multicast traffic to use different paths. For example, if a site has two links to its network provider, one of the links may be used for unicast only, and the other one only for multicast.

To provide for such flexibility in the configuration, the PIM-SM and PIM-DM specifications use the so called Multicast Routing Information Base (MRIB) for obtaining the RPF information. Typically, the MRIB may be derived from the unicast routing table, but some protocols such as MBGP may carry multicast-specific topology information. Furthermore, the MRIB may be modified locally in each site by taking into account local configuration and preferences. A secondary function of the MRIB is to provide routing metrics for destination addresses. Those metrics are used by the PIM-SM and PIM-DM Assert mechanism.

12.2 Configuring the MRIB

The XORP RIB module contains a table with the MRIB. That table is propagated to the PIM-SM module and is used by PIM-SM in the RPF computation. The MRIB table inside the RIB module is completely independent from the Unicast Routing Information Base (URIB) table. The URIB table is created from the unicast routes calculated by unicast routing protocols such as BGP, OSPF and RIP. The MRIB table is created similarly, but only by those protocols that are explicitly configured to add their routes to the MRIB. For example, if Multi-protocol BGP is enabled, then the BGP module will add multicast-specific routes to the MRIB.

Currently, XORP supports the following methods for adding routing entries to the MRIB:

- `Multi-protocol BGP`: The BGP module can be configured to negotiate multiprotocol support with its peers. Then, the BGP multicast routes will be installed in the MRIB. See Chapter 8 for information how to configure BGP.
- `Static Routes`: The Static Routes module can be used to configure static unicast and multicast routes. The unicast routes are added to the Unicast RIB, while the multicast routes are added to the MRIB. See Chapter 6 for information how to configure Static Routes.
- `FIB2MRIB`: If there are no unicast routing protocols configured in XORP to supply the MRIB routes, then the FIB2MRIB module can be used to populate the MRIB. If the FIB2MRIB module is enabled, it will register with the FEA to read the whole unicast forwarding table from the underlying system, and to receive notifications for all future modifications of that table. In other words, the FIB2MRIB task is to replicate the unicast forwarding information on that router into the MRIB. The FIB2MRIB module can be enabled by the following configuration statements: [1]

```
protocols {
  fib2mrib {
    disable: false
  }
}
```

12.3 Monitoring the MRIB

All operational commands for monitoring MRIB begin with `show route table`. This section describes those commands in details.

All RIB commands to view the RIB's inner tables have the following form:

`show route table {ipv4 | ipv6} {unicast | multicast} <protocol>`

The commands to view the MRIB have the following form:

`show route table {ipv4 | ipv6} multicast <protocol>`

The value of `<protocol>` has to be one of the following:

- `ebgp` to show eBGP MBGP routes.
- `fib2mrib` to show unicast routes for multicast extracted from kernel.
- `final` to show MRIB winning routes.

[1]Note that prior to XORP Release-1.1, the `enable` flag was used instead of `disable` to enable or disable each part of the configuration.

- `ibgp` to show iBGP MBGP routes.
- `static` to show MRIB static routes.

For example, the following command can be used to show the IPv4 FIB2MRIB routes:

```
Xorp> show route table ipv4 multicast fib2mrib
Network 10.2.0.0/24
    Nexthop := 0.0.0.0
    Metric := 65535    Protocol := fib2mrib    Interface := dc2    Vif := dc2
Network 10.3.0.0/24
    Nexthop := 0.0.0.0
    Metric := 65535    Protocol := fib2mrib    Interface := dc1    Vif := dc1
Network 10.4.0.0/24
    Nexthop := 10.3.0.2
    Metric := 65535    Protocol := fib2mrib    Interface := dc1    Vif := dc1
Network 10.5.0.0/24
    Nexthop := 10.2.0.4
    Metric := 65535    Protocol := fib2mrib    Interface := dc2    Vif := dc2
Network 10.6.0.0/24
    Nexthop := 10.2.0.1
    Metric := 65535    Protocol := fib2mrib    Interface := dc2    Vif := dc2
```

The final MRIB table can be shown with the following command: routes:

```
Xorp> show route table ipv4 multicast final
Network 10.2.0.0/24
    Nexthop := 10.2.0.2
    Metric :=     0    Protocol := connected   Interface := dc2    Vif := dc2
Network 10.3.0.0/24
    Nexthop := 10.3.0.1
    Metric :=     0    Protocol := connected   Interface := dc1    Vif := dc1
Network 10.4.0.0/24
    Nexthop := 10.3.0.2
    Metric := 65535    Protocol := fib2mrib    Interface := dc1    Vif := dc1
Network 10.5.0.0/24
    Nexthop := 10.2.0.4
    Metric := 65535    Protocol := fib2mrib    Interface := dc2    Vif := dc2
Network 10.6.0.0/24
    Nexthop := 10.2.0.1
    Metric := 65535    Protocol := fib2mrib    Interface := dc2    Vif := dc2
```

Chapter 13

SNMP

13.1 Terminology and Concepts

SNMP (Simple Network Management Protocol) is a mechanism for managing network and computer devices. SNMP uses a manager/agent model for managing the devices. The agent resides in the device, and provides the interface to the physical device being managed. The manager resides on the management system and provides the interface between the user and the SNMP agent. The interface between the SNMP manager and the SNMP agent uses a Management Information Base (MIB) and a small set of commands to exchange information.

The MIB contains the set of variables/objects that are managed (*e.g.,* MTU on a network interface). Those objects are organized in a tree structure where each object is a leaf node. Each object has its unique Object IDentifier (OID). There are two types of objects: `scalar` and `tabular`. A scalar object defines a single object instance. A tabular object defines multiple related object instances that are grouped in MIB tables. For example, the uptime on a device is a scalar object, but the routing table in a router is a tabular object.

The set of commands used in SNMP are: GET, GET-NEXT, GET-RESPONSE, SET, and TRAP. GET and GET-NEXT are used by the manager to request information about an object. SET is used by the manager to change the value of a specific object. GET-RESPONSE is used by the SNMP agent to return the requested information by GET or GET-NEXT, or the the status of the SET operation. The TRAP command is used by the agent to inform asynchronously the manager about the occurrence of some events that are important to the manager.

Currently there are three versions of SNMP:

- `SNMPv1`**:** This is the first version of the protocol. It is described in RFC 1157.
- `SNMPv2`**:** This is an evolution of the first version, and it adds a number of improvements to SNMPv1.
- `SNMPv3`**:** This version improves the security model in SNMPv2, and adds support for proxies.

13.2 Configuring SNMP

NOTE: Just before Release-1.1 we noticed that the SNMP support is broken, hence it should not be

enabled. This will be fixed immediately after the release.

Before configuring SNMP on XORP, you must make sure that SNMP support is compiled. For example, when running `./configure` in the top-level XORP directory, you have to supply the `--with-snmp` flag:

```
./configure --with-snmp
```

13.2.1 Configuring Net-SNMP

XORP itself does not implement the SNMP protocol and requires an external SNMP implementation for that. Currently, XORP supports only Net-SNMP (see `http://www.net-snmp.org`) as such implementation. Before configuring SNMP in XORP, you must take the following steps to configure your Net-SNMP agent to run with XORP:

- You need Net-SNMP version 5.0.6 or greater.
- You must make `libnetsnmpxorp.so` accessible to your runtime loader. Depending on your system, that requires one of the following:
  - Copy `libnetsnmpxorp.so` to your library directory (typically `/usr/local/lib`).
  - Set a linker environment variable (typically `LD_LIBRARY_PATH`) to point to the directory where the library is.
- To avoid opening security holes, we recommend allowing only SNMPv3 authenticated requests. If you want to create a secure user, execute the command `net-snmp-config --create-snmpv3-user`. These are the settings that match the provided `snmp.conf` file inside the `${XORP}/mibs/snmpdscripts/` directory:

User	Pass phrase	Security level
privuser	I am priv user	authPriv

  You must create at least one user if you want to be able to access the SNMP agent.
- `snmpd` can only respond to XRLs after `xorp_if_mib_module` has been loaded. Adding the following line to the file `snmpd.conf` (by default located in `/usr/local/share/snmp`) will preload this module when `snmpd` is started:

  ```
  dlmod xorp_if_mib_module <absolute path full filename>
  ```

  For example:

  ```
  dlmod xorp_if_mib_module /usr/local/xorp/mibs/xorp_if_mib_module.so
  ```

13.2.2 Configuration Syntax

```
protocols {
  snmp {
    targetname: text
    mib-module text {
      abs-path: text
      mib-index: uint
    }
  }
}
```

`protocols`: this delimits the configuration for all routing protocols in the XORP router configuration. It is mandatory that SNMP configuration is under the `protocols` node in the configuration.

`snmp`: this delimits the SNMP configuration part of the XORP router configuration.

`targetname`: this is the name for this instance of SNMP. It defaults to "`xorp_if_mib`", and it is not recommended that this default is overridden under normal usage scenarios.

`mib-module`: this specifies the MIB module to configure. It should be set to the MIB module file name (without the file name extension).

For each MIB, the following parameters can be configured:

`abs-path`: this is the absolute path to the module file with the MIB to load.

`mib-index`: this is the MIB index. It is set internally by XORP when a MIB module is loaded, and should not be set in the XORP configuration.

Below is a sample SNMP configuration that configures a BGP MIB:

```
protocols {
  snmp {
    mib-module bgp4_mib_1657 {
      abs-path: "/usr/local/xorp/mibs/bgp4_mib_1657.so"
    }
  }
}
```

13.3 Using SNMP to Monitor a Router

Currently (April 2005) XORP does not provide SNMP-related operational commands.

However, there are few client-side scripts that can be used to experiment with the SNMP agent:

- The scripts are in the `${XORP}/mibs/snmpdscripts/` directory, and they use the client-side Net-SNMP tools to communicate with the agent. They rely on file `snmp.conf` in the same directory to provide valid default values for the SNMP version to use, the user, community and security level. If your agent was configured with the default security user suggested in Section 13.2.1, you should copy the `${XORP}/mibs/snmpdscripts/snmp.conf` file to `${HOME}/.snmp/snmp.conf`. Otherwise, you'll have to create your own `snmp.conf` so it matches your settings.

- You must make XORP textual MIB files (`${XORP}/mibs/textual/*.txt`) accessible to the Net-SNMP command line tools. Either set the `MIBDIRS` environment variable (`man snmpcmd(1)`) to point to that directory or copy those files to your MIBS directory (default is `/usr/local/share/snmp/mibs`). For instance (if `sh` is the login shell):

  ```
  export MIBDIRS=+/usr/local/xorp/mibs/textual
  ```

- You must tell Net-SNMP about specific MIB modules that you will be using. The `MIBS` environment variable can be used for that purpose. For BGP4-MIB you would do (if `sh` is the login shell):

  ```
  export MIBS=+BGP4-MIB
  ```

Chapter 14

User Management

Currently, XORP does not support user management. If XORP is running on a multi-access UNIX system, an user who needs an access to XORP must have an UNIX account on that system. See Chapter 16 for information about user management if XORP is running from a LiveCD.

Any user on a multi-access UNIX system can start and use `xorpsh` in operational mode. However, only users that belong to the "xorp" UNIX group can run `xorpsh` in configurational mode.

In the future XORP will provide better user access control mechanism, and will provide a mechanism for user management (*e.g.,* adding and deleting users that can access XORP, etc).

Chapter 15

Diagnostics and Debugging

15.1 Debugging and Diagnostic Commands

XORP supports severail operational commands in `xorpsh` that can be used for debugging or diagnostics purpose.

The `ping <host>` command can be used to test if a network host responds to ICMP ECHO_REQUEST packets:

```
Xorp> ping 10.3.0.2
PING 10.3.0.2 (10.3.0.2): 56 data bytes
64 bytes from 10.3.0.2: icmp_seq=0 ttl=64 time=0.281 ms
64 bytes from 10.3.0.2: icmp_seq=1 ttl=64 time=0.244 ms
64 bytes from 10.3.0.2: icmp_seq=2 ttl=64 time=0.302 ms
64 bytes from 10.3.0.2: icmp_seq=3 ttl=64 time=0.275 ms
Xorp> ping 10.3.0.2
Command interrupted!
```

The `ping` command can be interrupted by the `Ctrl-C` key combination.

The `traceroute <host>` command can be used to print the route packets take to a network host:

```
Xorp> traceroute 10.4.0.2
traceroute to 10.4.0.2 (10.4.0.2), 64 hops max, 44 byte packets
 1  xorp3-t2 (10.3.0.2)  0.451 ms  0.366 ms  0.384 ms
 2  xorp7-t0 (10.4.0.2)  0.596 ms  0.499 ms  0.527 ms
```

The `traceroute` command can be interrupted by the `Ctrl-C` key combination.

The `show host` commands can be used to display various information about the host itself.

The `show host date` command can be used to show the host current date:

```
Xorp> show host date
Mon Apr 11 15:01:35 PDT 2005
```

The `show host name` command can be used to show the host name:

```
Xorp> show host name
xorp2
```

The `show host os` command can be used to show details about the host operating system:

```
Xorp> show host os
FreeBSD xorp2 4.9-RELEASE FreeBSD 4.9-RELEASE #0: Wed May 19 18:56:49 PDT 2004
   atanu@xorpc.icir.org:/scratch/xorpc/u3/obj/home/xorpc/u2/freebsd4.9.usr/src/sys/XORP-4.9 i386
```

Chapter 16

XORP Live CD

The XORP Live CD is a bootable CD for x86 PCs. The Live CD serves a number of purposes:

- It's an easy way to try out XORP without needing to compile anything or reformat the disk on your PC.
- It's a quick way to get a relatively secure router on demand.
- It's a great tool for a student lab session, requiring no installation.

See the XORP Web site (`http://www.xorp.org/`) for information how to download the lastest version of the XORP LiveCD ISO image. Once you've downloaded the CD image, you will need to burn it using a CD-R or CD-RW drive. For example, in case of FreeBSD you can simply run:

```
burncd -f /dev/acd0c -e data LiveCD.iso fixate
```

See the XORP Web site for some URLs with instructions on how to burn CD images on other systems.

16.1 Running the Live CD

To boot from the Live CD, your PC needs to have the CD-ROM device set as the primary boot device. If this is not already the case, you will need to modify the settings in the BIOS. The boot order should along the lines of:

1. CD drive.
2. Floppy Disk.
3. Hard Disk.

The order of the floppy and hard disk are unimportant, just so long as they're after the CD drive in the boot order. This is usually pretty easy to change in the BIOS - you might want to make a note of the original boot order in case you want to switch it back afterwards. Typically to change BIOS settings, you hold down Delete or F2 (depending on your PC) just after you restart your PC.

If you want the router to store any configuration changes you have made when it is rebooted, you'll also need a floppy disk, but you can try the Live CD without this.

Then reboot the PC. The PC should boot from the CD. Normally it will display a low resolution XORP logo for 30 seconds to a minute while booting completes. Sorry - there's no progress bar to let you know anything is happening.

If you've got a floppy in the floppy drive, and you've done this before, then the XORP configuration will be copied into the memory filesystem, along with passwords, sshd keys, etc. Then the XORP routing protocols will be started.

If there's no floppy in the drive, or it doesn't have the files on it that XORP expects, then a simple interactive script will run to allow you to configure passwords and decide which network interfaces you want XORP to use.

16.2 Starting XORP the First Time

The startup script that runs the first time you run XORP is quite simple. If there's no floppy in the floppy drive, or it's not DOS-formatted, you'll be presented with a warning similar to the one in Figure 16.1.

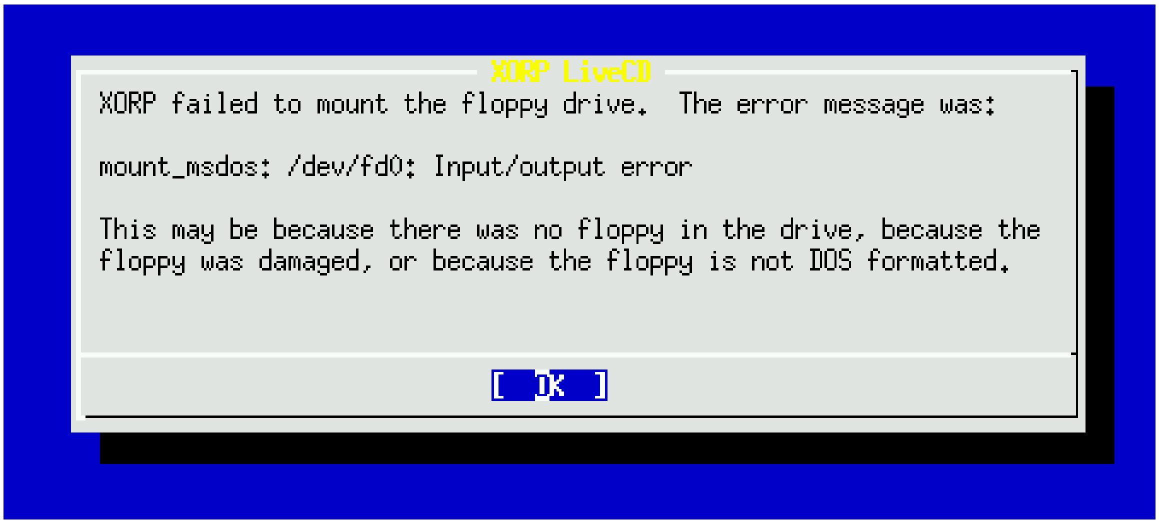

Figure 16.1: LiveCD missing floppy-related warning

Hit enter, and you'll be given the choices shown in Figure 16.2.

Use the cursor keys to move up and down to choose an option, and hit enter.

If you hadn't got a floppy in the drive, you can add one now, and select 1.

If your floppy is not DOS formatted, you can reformat it (erasing all the data on it) by selecting 3.

If you don't have a floppy to hand, you can continue by selecting 2, but you won't be able to preserve any configuration changes you make later.

If you now have a blank writable DOS formatted floppy in the floppy drive, you'll get the notice shown in Figure 16.3.

Figure 16.2: LiveCD floppy-related menu

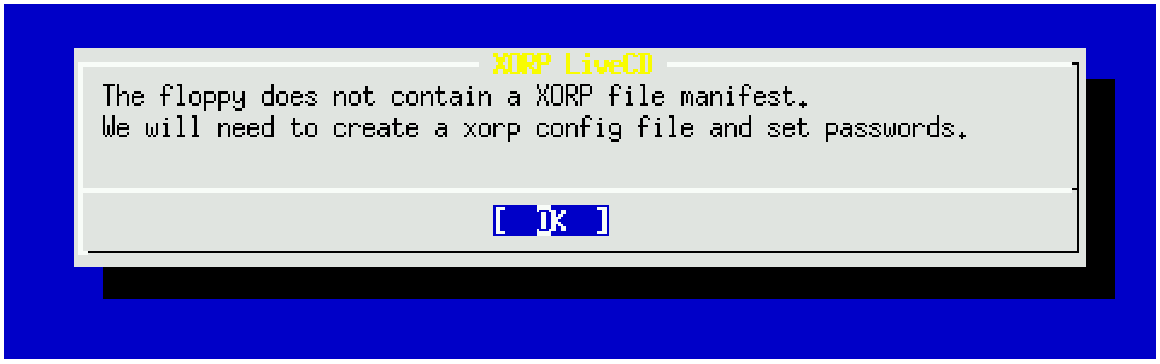

Figure 16.3: LiveCD floppy-related message

Hit Enter, and you will be prompted to enter the root password for the FreeBSD system. This will allow you to login to the machine as the superuser to diagnose any problems, or to see how XORP works behind the scenes.

Next you will be prompted to enter the password for the ”xorp” user account. On a normal XORP router, you might have many user accounts for the different router administrators, but on the Live CD we just create one user called ”xorp”. Please do enter a reasonable password, as this user will be able to login over the network using the ssh secure shell and this password.

Finally you will be prompted as to which network interfaces you wish XORP to manage. These interfaces will show up in the default XORP configuration file, ready to have IP addresses assigned. The menu looks like the one shown in Figure 16.4.

Typically you will only want XORP to manage Ethernet interfaces and the loopback interface from the Live CD at this stage, because currently XORP has no built-in support for dial-up links. Move up and down using the cursor keys, and hit space to select or unselect an option (an ”X” implies the option is selected). When you are finished, hit Tab, to select the ”OK” button, and hit Enter.

That’s it. XORP will now finish booting.

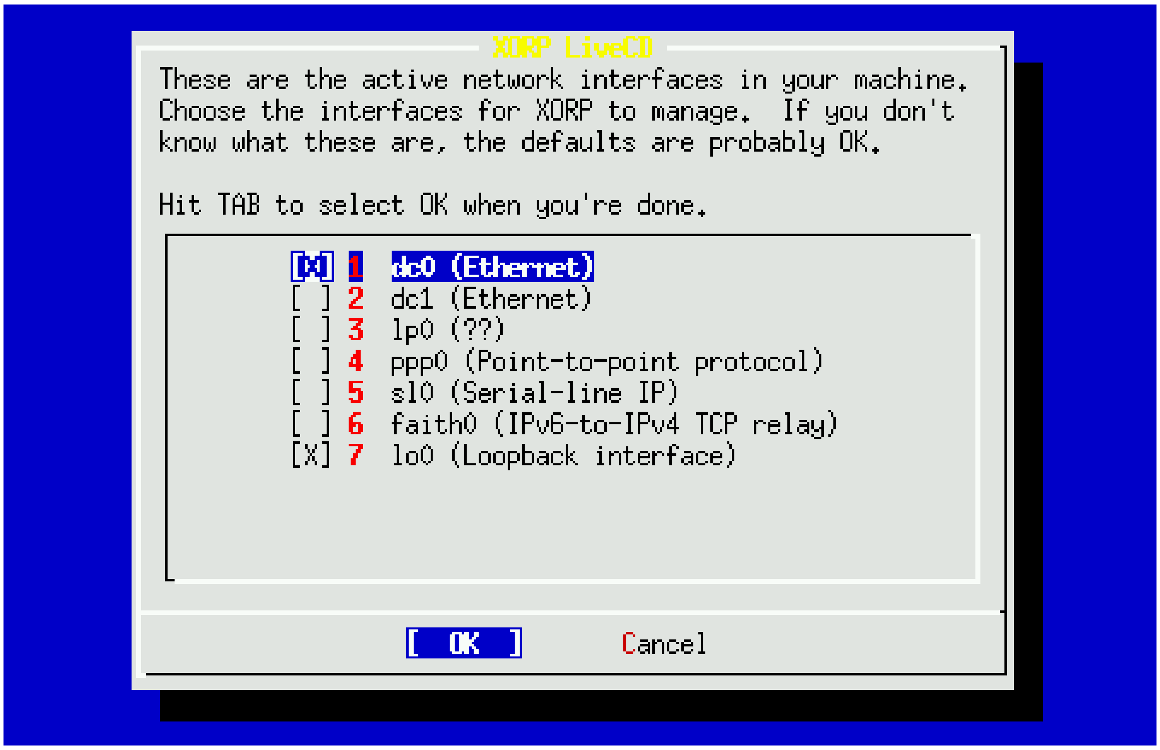

Figure 16.4: LiveCD network interfaces menu

Once XORP has finished booting, you will be presented with a login prompt, and you can login to XORP as the ”xorp” user with the password you have chosen, and interact with the XORP command line interface to complete the configuration, assign IP addresses, etc.

16.3 Saving Config

The location of the router configuration file used by XORP can be set using command line parameters, so different XORP systems might choose to use a different location for this file. On the Live CD, the configuration file is stored in `/etc/xorp.cfg`.

If you change the router configuration using the XORP shell, and want to save it, you need to enter the following in configuration mode:

```
XORP> save /etc/xorp.cfg
```

If you save to any other location, the file will still be preserved on the floppy, but will not be loaded automatically the next time XORP reboots.

16.4 Debugging

The Live CD includes two versions of the XORP system binaries. The normal version is mounted in a memory filesystem in `/usr/local/xorp</B>`. This version has had the debugging systems stripped so that the binaries are small enough to reside in a memory filesystem. This allows them to load quickly, and to run on a PC with less memory.

If you need a debugging version, you can run the following command:

```
umount /usr/local/xorp
```

A second copy of `/usr/local/xorp` with debugging binaries resides on the CD, and is revealed when the memory filesystem is unmounted. These binaries are rather large, and load slowly, so don't use them unless you really need them. Using them rather assumes you know how XORP works internally, so is beyond the scope of this tutorial.

16.5 Interface Naming

If you're used to Linux, you may be surprised that FreeBSD names it's Ethernet interfaces with names like `fxp0`, `fxp1`, `dc0` and `xl3`, rather than `eth0`, `eth1`, etc. The advantage is that you can tell exactly what the device driver is that's being used, and that if you know you have one Intel 10/100 and one DEC Tulip in the machine, you know they'll be called `fxp0` and `dc0`, no matter which PCI slot they're in. The disadvantage is that it's more confusing for beginners who don't want to know this detail.

Some people get religious about such things. We don't - this just reflects the underlying operating system's naming convention. If you ran XORP on Linux, you'd see `eth0`, etc.