XORP PIM-SM Routing Daemon

Version 0.1

XORP Project
International Computer Science Institute
Berkeley, CA 94704, USA
feedback@xorp.org

December 11, 2002

1 Introduction

1.1 Overview

This document provides an overview of the XORP PIM-SM [2] Routing Daemon. It is intended to provide a starting point for software developers who wish to modify this software.

A router running PIM-SM interacts with other PIM-SM routers and multicast members, computes the multicast routing state, and installs the corresponding multicast forwarding state in the multicast forwarding engine.

The chosen architecture for our PIM-SM implementation emphasizes on correctness and extensibility rather than high performance or minimal memory footprint. PIM-SM is a fairly complicated protocol, therefore it is very important that the implementation follows closely the protocol specification. Otherwise, premature optimization or "cutting corners" might introduce problems that are difficult to find. Only after the implementation is well tested, we should try to optimize those parts of the implementation that should prove to be a bottleneck.

Currently (December 2002), the PIM-SM implementation is based on the specification in the following documents:

- `draft-ietf-pim-sm-v2-new-05.{ps,txt}` (The core PIM-SM specification).
- `draft-ietf-pim-sm-bsr-02.{ps,txt}` (The Bootstrap mechanism).

The only major features not implemented yet are SSM support and security.

1.2 Acronyms

Acronyms used in this document:

- **MFC**: **M**ulticast **F**orwarding **C**ache: another name for an entry in the multicast forwarding engine (typically used on UNIX systems).
- **MFEA**: **M**ulticast **F**orwarding **E**ngine **A**bstraction
- **MLD/IGMP**: **M**ulticast **L**istener **D**iscovery/**I**nternet **G**roup **M**anagement **P**rotocol

- **MRIB**: **M**ulticast **R**outing **I**nformation **B**ase
- **PIM-SM**: **P**rotocol **I**ndependent **M**ulticast–**S**parse **M**ode
- **RIB**: **R**outing **I**nformation **B**ase

1.3 PIM-SM Design Architecture Overview

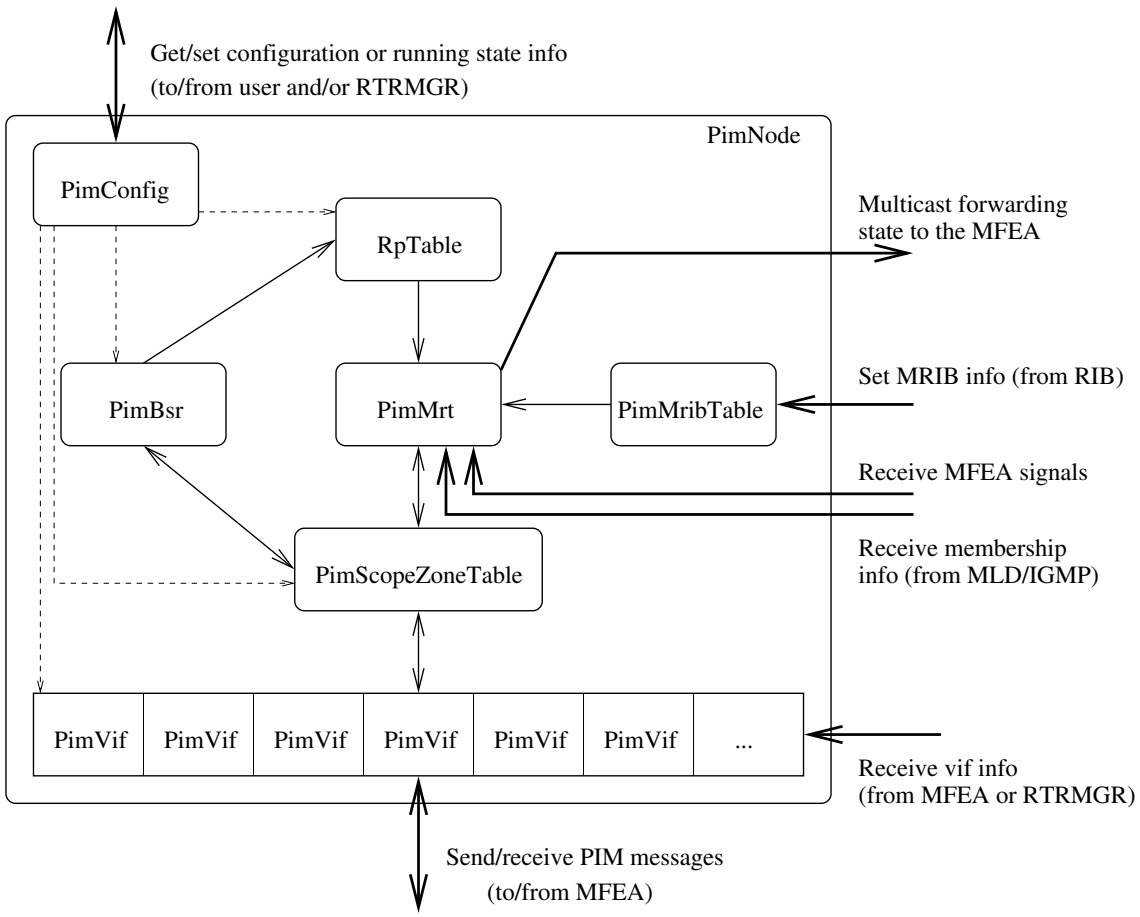

Figure 1: PIM-SM design overview

Figure 1 gives a general overview of the PIM-SM components. For each component there is a C++ class with exactly the same name. The main components are briefly described below:

- **PimNode:** a representation of a single PIM-SM routing unit (*e.g.,* a virtual PIM-SM router). Typically, there would be a single PimNode per PIM-SM router.
- **PimVif:** PIM-specific virtual interface that is used for sending and receiving PIM control messages.
- **PimSconeZoneTable:** a table that contains information about scoped zones.
- **PimMrt:** PIM-specific multicast routing table.

- **PimBsr:** the PIM-Bootstrap mechanism unit.
- **RpTable:** the table with the PIM-SM RP information.
- **PimMribTable:** the table with the MRIB information.
- **PimConfig:** contains PIM-specific configuration.

Those components are described in details in Section 2. For information about the interaction between the PIM-SM and other modules see [1].

2 Components Description

2.1 PimNode Description

PimNode is a representation of a single PIM-SM routing unit (*e.g.,* a virtual PIM-SM router). Typically, there would be a single PimNode per PIM-SM router. However, in some cases a PIM-SM router may have more than one routing unit. For example, it could have one PimNode for IPv4, and another one for IPv6 multicast routing. Further, if we want to run PIM-SM in a simulation environment, each PIM-SM router within that simulation will be represented by a single PimNode.

From a developer's point of view, PimNode contains the front-end interface to interact with the PIM-SM routing unit, as well as all the state related to that unit. For example, PimNode contains the methods to start/stop or configure PIM-SM, or to send/receive PIM control messages to/from the routing unit. Those methods are described in the following files:

- `pim/pim_node.hh`
- `libproto/proto_node.hh`
- `libproto/proto_unit.hh`

PimNode itself does not implement the mechanisms to communicate with other routing units (*e.g.,* to send or receive control packets to/from the network), or to perform other PIM-independent operations such as installing multicast forwarding entries in the multicast forwarding engine. Those mechanisms are outside the scope of PimNode, and must be implemented separately.

PimNode contains several pure virtual methods (*e.g.,* `join_multicast_group()` that is used to join a multicast group on an interface) that must be implemented by a class that inherits PimNode. For example, XrlPimNode is a class that uses PimNode as a base class, and that implements XRL-based communication mechanisms between PimNode and other XORP components such as the MFEA and MLD/IGMP modules.

By default, PimNode is disabled; therefore it must be explicitly enabled, and then started.

2.2 PimVif Description

PimVif is a PIM-specific virtual interface that is used for sending and receiving PIM control messages. It includes the methods for processing and composing PIM control messages, as well as various state per interface (*e.g.,* the set of state for PIM-SM neighbors on an interface).

Typically, there would be one PimVif virtual interface for each interface on the router: physical interfaces, tunnels, and the loopback interface. In addition, there will be one special PimVif virtual interface: the PIM Register virtual interface that is used for sending and receiving PIM Register messages. Not all virtual

interfaces will be used by PIM; for example, all interfaces that are not multicast capable, and the loopback interface will be ignored for multicast routing.

Typically, from developer's point of view, all interaction with PimVif would be through PimNode [1].

The public interface for PimVif contains the methods to manipulate a virtual interface. Those methods are to start/stop/enable/disable a virtual interface, or to configure it. Those methods are described in the following files:

- `pim/pim_vif.hh`
- `libxorp/vif.hh`
- `libproto/proto_unit.hh`

PimVif contains state such as PIM Hello related information. Also, all of the PIM-specific methods for parsing or constructing PIM control messages when a PIM packet is received or sent are implemented as methods to PimVif. The parsing or construction of each message type is implemented in a separate file with a name prefix of `pim_proto`. For example, `pim_proto_cand_rp_adv.cc` implements sending and receiving of PIM Candidate-RP-Advertisement messages. The handing of other message types is implemented in similarly named files.

By default, each PimVif is disabled; therefore it must be explicitly enabled, and then started.

2.3 PimScopeZoneTable Description

PimSconeZoneTable is a table that contains information about scoped zones. There is one table per PimNode. This table is used to check whether various control messages are allowed to be send or accepted on specific interfaces [2].

By default, the PimScopeZoneTable is empty; *i.e.,* there are no scoping restrictions.

2.4 PimMrt Description

PimMrt is the PIM-specific multicast routing table. It is the central and most important component: its state is modified by the PIM control messages, and the output of it is the multicast forwarding state information that is installed in the multicast forwarding engine.

The multicast routing table is composed of four tables. Each table contains PimMre entries (described in file `pim/pim_mre.hh`):

- (*,*,RP) multicast routing table. This table contains all (*,*,RP) multicast routing entries [3]. For simplicity of implementation, this table contains an (*,*,RP) entry for each RP in the RpTable, even if no (*,*,RP) Join messages for that RP were received. The iterator for this table returns the entries ordered by their RP address: the numerically smallest addresses first. Note that all PimMre entries in this table have the RP address as the source address, and group address of zero (*e.g.,* `IPvX::ZERO()`).
- (*,G) multicast routing table. This table contains all (*,G) multicast routing entries. Each entry in that table contains a pointer to the corresponding (*,*,RP) entry for that group, or NULL if the group has

[1] Note that on few occasions in the current implementation (December 2002) XrlPimNode for simplicity uses direct access to PimVif.

[2] Note that in the current implementation (December 2002) the PimScopeZoneTable is used only for PIM Bootstrap messages. In the future, the scope zone information would be used for other control messages as well.

[3] (*,*,RP) entry is an entry that matches all multicast groups that use one specific RP.

no RP yet. The iterator for this table returns the entries ordered by their group address: the numerically smallest addresses first. Note that all PimMre entries in this table have a source address of zero (*e.g.,* `IPvX::ZERO()`).

- (S,G) multicast routing table. This table contains all (S,G) multicast routing entries. Each entry in that table contains a pointer to the corresponding (*,G) entry for that group, or NULL if there is no (*,G) entry. It also contains a pointer to the corresponding (S,G,rpt) entry if such exists (seen below). There are two iterators for this table: entries ordered by source address first, and entries ordered by group address first.

- (S,G,rpt) multicast routing table. This table contains all (S,G,rpt) multicast routing entries. Each entry in that table contains a pointer to the corresponding (*,G) entry for that group (by definition, there must be (*,G) entry to have (S,G,rpt) entry as well). It also contains a pointer to the corresponding (S,G) entry if such exists. There are two iterators for this table: entries ordered by source address first, and entries ordered by group address first.

For simplicity of implementation, currently (December 2002) PimMrt contains one more table: PimMrtMfc PIM-specific table with Multicast Forwarding Cache entries (in the future, this table may be moved out of PimMrt to PimNode) composed of PimMfc entries. This table contains all entries that have been installed in the multicast forwarding table in the multicast forwarding engine. Currently (December 2002), those entries are source-group-specific, and are installed "on-demand" (*i.e.,* only if there is an active source for some group). In the future, group-specific entries may be supported as well (assuming that that multicast forwarding engine supports (*,G) multicast forwarding entries).

In addition to the above tables, PimMrt contains a mechanism for tracking dependencies among the PimMre and PimMfc entries, as well as the PimMre and PimMfc dependencies on external state such as the RP set or the MRIB information. For example, if the MRIB for a specific network prefix changes, then all PimMre and PimMfc entries that depend on that network prefix must be updated accordingly. A single change may trigger a number of operations that must be performed on a number of entries, therefore we need to carefully track state dependency. Below is a summary of some of the events that may trigger processing actions in the PimMrt:

- RP-set change: *e.g.,* if there is any change to the RP-set that affects the group-to-RP mapping.

- MRIB change: any change in the underlying unicast routing that affects the Reverse-Path Forwarding information toward an RP or a source.

- Next-Hop PIM neighbor change: any change to the set of PIM neighbors that may affect a Next-Hop PIM Router.

- Reception of a PIM Join/Prune message.

- Reception of a PIM Assert message.

- Add/deletion of a local multicast member.

- Change in the Designated Router on an interface.

- Change in the IP address or IP subnet on an interface.

- Start or stop a virtual interface.

- Addition or deletion of a PimMre entry.

A complete list of all input events that may trigger actions is in file `pim/pim_mre_track_state.hh` (see the `input_state_t INPUT_STATE_*` events).

In some cases, keeping track of the entries that need to be processed for a given input event is relatively simple. For example, if the MRIB for a network prefix changes, processing all (S,G) PimMre entries that might be affected can be done by using the source-first iterator for the (S,G) multicast routing table, and iterating over all (S,G) PimMre entries whose source address matches that network prefix. In other cases however we cannot use those table iterators. For example, if an RP is deleted, we need to process all corresponding (*,G) entries that match to that RP, and reassign them to a new RP. In that case, to keep track of the dependencies between the RP and the (*,G) entries, each RP entry in the RpTable contains a list of PimMre entries that match to that RP. Similarly, each PimNbr entry (an entry that contains information about a PIM neighbor) contains a list of all PimMre entries that use that PIM neighbor as the Next-Hop Router toward the RP or the source.

The dependency tracking mechanism needs to solve another problem: for each input event, what are the operations and their ordering that need to be performed on some of the PimMre and PimMfc entries. The solution for this is to enumerate all possible input events and output operations, and to compute in advance a table that for each given input event returns a list of the ordered output operations that need to be performed.

If there are just few input events and output operations, it might be possible to create such table manually. However, currently (December 2002) there are 47 input events and 70 output operations, therefore manually creating such table is not feasible. The solution is on startup to automatically compute this table based on a set of rules about the various state dependencies as defined in the PIM-SM spec. Those state dependencies are derived from the macros in the PIM-SM protocol specification. For example, the specification document contains macros like:

```
pim_include(S,G) =
    { all interfaces I such that:
      ( (I_am_DR( I ) AND lost_assert(S,G,I) == FALSE )
        OR AssertWinner(S,G,I) == me )
       AND  local_receiver_include(S,G,I) }
```

Then, the corresponding state dependency rule in the implementation is:

```
void
PimMreTrackState::track_state_pim_include_sg(list<PimMreAction> action_list)
{
    track_state_i_am_dr(action_list);
    track_state_lost_assert_sg(action_list);
    track_state_assert_winner_sg(action_list);
    track_state_local_receiver_include_sg(action_list);
}
```

In other words, if the value of `lost_assert(S,G,I)` for example changes, then the value of `pim_include(S,G)` must be recomputed. However, we may have some state dependency rules for `lost_assert(S,G,I)` itself, hence if we combine all state dependency rules, we can represent the dependencies with a collection of uni-directional graphs. Then, to create the list of actions for each input entry, we need to consider all paths from the graph node that corresponds to that input entry to all reachable output actions. The uni-directional graphs creation and the extraction of the lists of actions for each input entry is performed once on startup. The result lists are saved internally inside PimMrt and used during processing of input actions.

Finally, the last major problem that the dependency tracking mechanism needs to solve is how to process a large number of entries triggered by a single event without stopping other processing. This problem arises because the implementation is single-threaded, therefore if processing a single event takes too long, the rest of the pending events may be processed too late (*e.g.,* if the periodic sending of PIM Hello messages is delayed for too long, the PIM neighbors may timeout this router). The solution of this problem is to voluntarily suspend the processing if it is taking too long, save the necessary state to continue later, and return control to the control loop which handles all events. Typically, the processing of some event may take too long if there is a large number of PimMre or PimMrt entries that need to be processed (for example, thousands of (*,G) entries if the RP for those entries changes). In that case, we use "time-slices" to compute how long has taken after processing each (*,G) entry. If the elapsed time is above a threshold (default to 100ms as of December 2002), we save the appropriate state to continue the processing later (*e.g.,* in the above example the address of the next multicast group to process).

All dependency tracking processing and time-slicing uses PimMreTask entries to keep the appropriate state. There is a single list of PimMreTask entries per PimNode, and the list is FIFO: new tasks are added to the end of the lists, and the task at the front of the list is processed until it is completed (*e.g.,* within one or several time-slices).

2.5 PimBsr Description

PimBsr is the PIM-Bootstrap mechanism unit. It implements the Bootstrap mechanism as described in [3]. There is one PimBsr unit per PimNode. The main purpose of the PimBsr is to run the Bootstrap mechanism, and to update accordingly the RpTable with the current RP-set.

The data contained in PimBsr is organized as described below. PimBsr keeps two lists of BsrZone entries: one list for the active BSR zones, and another list for the locally configured Cand-BSR zones and/or Cand-RP information. On startup, the active BSR zones list is empty. If the node is configured as a Candidate-BSR and/or a Candidate-RP, this information will be added to the second list; otherwise that list is also empty.

Each scope zone is identified by a scope zone ID, and a flag. The flag, if true, indicates that this is scoped zone, otherwise the zone is non-scoped. The scone zone ID is the network prefix address that corresponds to that zone. By definition, scoped zones cannot overlap, therefore it is not permitted to configure the PimBsr with overlapping scoped zones and/or to accept Bootstrap messages with scoped zones that overlap. If the zone is non-scoped, then the scope zone ID is set to the multicast base prefix address (*i.e.,* 224.0.0.0/4 for IPv4 or FF00::/8 for IPv6).

Each BsrZone contains information about the current BSR for that zone, and a list of BsrGroupPrefix entries for that zone. Each BsrGroupPrefix corresponds to a multicast group prefix within that zone that has Candidate-RPs, and contains the list of BsrRp entries for each Candidate-RP for that prefix.

All information from the Bootstrap and Candidate-RP messages is kept in the above data structures. Further, those structures are used to keep various timers such as to timeout Candidate RPs or the current BSR. If the RP-set is changed after receiving a Bootstrap message, or after a timeout, the RpTable is updated accordingly.

2.6 RpTable Description

RpTable is the table that contains the current RP-set. There is one table per PimNode. This table is updated by PimBsr if the RP-set is propagated through the Bootstrap mechanism, or by PimConfig if the RP-set is configured manually.

RpTable contains a list of all RPs with one PimRp entry per RP per multicast group prefix. If we need to compute the RP for a given group, we just scan the whole list to find the RP that this group maps to.

Typically, the list of RPs will be relatively short, therefore for simplicity we scan the whole list. If the overhead becomes too large, then it is possible to optimize the scan by grouping the Candidate-RPs for each group prefix, and considering only the Candidate-RPs with the highest priority.

If the RP-set is modified, then all affected PimMre and PimMfc entries must be updated accordingly. For this reason, each PimRp entry contains lists of the PimMre and PimMfc entries that map to that RP. If this RP is removed, then all entries on those lists are re-mapped to the new RPs for each group. Note that in general after the re-mapping some of the groups may map to a different RP, therefore we must process all entries one-by-one. This is achieved by scheduling a PimMreTask by the PimMrt, that takes the appropriate dependency actions for each entry.

The RpTable may contain one special PimRp entry with address of all-zeroes. This entry is used to keep the lists of all PimMre and PimMfc entries that have no RP yet. If a new RP is added to the RpTable, then all entries that have no RP yet are processed to find if some of them may map to this new RP. Those who do map are moved to the appropriate list on the new RP.

2.7 PimMribTable Description

PimMribTable is the table with the MRIB information. The MRIB is used to compute the Reverse-Path Forwarding information toward the RPs (needed by the (*,*,RP), (*,G) and (S,G,rpt) state), and toward each active multicast sender (needed by the (S,G) state). This information contains the next-hop router address and the interface toward that router, the routing metric and the metric preference:

```
// Reverse-Path Forwarding information (MRIB payload entry)
class Mrib {
    IPvXNet     _dest_prefix;          // The destination prefix address
    IPvX        _next_hop_router_addr; // The address of the next-hop router
    uint16_t    _next_hop_vif_index;   // The vif index to the next-hop router
    uint32_t    _metric;               // The routing metric to the dest.
    uint32_t    _metric_preference;    // The routing metric preference
                                       // to the destination
};
```

The MRIB information is obtained from the RIB module [4]; if the RIB changes, the PimMribTable is updated as well. Examples when the MRIB information may change are: the unicast routing changes the next-hop router address toward a destination, local configuration changes some of the routing preference metrics, or local interface configuration changes the virtual interface and/or the next-hop router toward a destination.

An update to the PimMribTable may affect a number of PimMre and PimMfc entries in the PimMrt table. The update of the affected entries is handled by the dependency-tracking and time-slice processing mechanism implemented by the PimMrt table. Note that we do not need to link all PimMre and PimMfc entries into lists of entries that depend on a particular entry in the PimMribTable. The reason is because the dependency is implied by the network prefix address covered by an entry in the PimMribTable, and the RP or source address of a given PimMre or PimMfc entry.

2.7.1 MRIB Changes Update

In general, there are two mechanisms to inform the PIM-SM module about MRIB changes:

[4] Currently (December 2002), the information is received from the MFEA instead.

- *Filtering at the PIM-SM module:* Whenever there are any changes about the MRIB information kept inside the RIB module, the RIB module informs the PIM-SM module about the changes. Then the PIM-SM module processes those changes to find whether they would affect in any way the current multicast routing.

- *Filtering at the RIB module:* The PIM-SM module "registers" in advance with the RIB module about the particular destination addresses/prefixes it is interested at, and only if the MRIB information about any of those registered destinations is changed, the RIB module informs the PIM-SM module about it.

The basic difference between the above two methods is where we move the complexity about the MRIB changes: at the RIB side, or the PIM-SM side. Some other differences are:

- If we perform filtering at the PIM-SM module, and if the unicast routing is changing quite rapidly, and if most of those changes do not affect the PIM-SM module, this will add unnecessary overhead to the communication from the RIB to the PIM-SM module.

- If we perform filtering at the RIB module, and if there is a large number of destinations the PIM-SM module needs to be informed about, registering all of those destinations at the RIB may introduce an "explosion" of communication from the RIB to the PIM-SM module if there is a change in the routing information about a large number of destinations.

- If we perform filtering at the PIM-SM module, the implementation may require the PIM-SM to keep a local (simplified) copy of all the RIB information, therefore it may increase notably the memory usage. This copy however can be used to perform the proper comparison and modification whenever MRIB update is received from the RIB module.

It may be possible to use some hybrid methods of propagating the MRIB changes from the RIB module to the PIM-SM module, but based on the above comparison it seems that *filtering at the PIM-SM module* is the simpler and more appropriate solution. Therefore, the RIB module needs to inform the PIM-SM module whenever the MRIB information for any destination prefix is changed. The simplest solution for the RIB module would be whenever any entry is changed, the RIB module would "dump" all RPF entries. This however may increase the communication overhead, and may complicate additionally the PIM-SM module. A better solution would be if the RIB modules sends only atomic updates of the RPF information to the PIM-SM module. For example, a single message would contain all affected entries: *e.g.,* a list of *MRIB_ADD* and *MRIB_DELETE* commands, and the particular ordering of those entries would specify also the order the PIM-SM module should apply them.

Note that as we mentioned earlier, currently (December 2002), the MRIB information is received from the MFEA, which reads directly the unicast forwarding table from the (UNIX) kernel, and PIM-SM keeps a local copy of the whole table. The MFEA periodically reads (*e.g.,* every 10 seconds) the unicast forwarding table, and if there is any change, it sends the changes to all interested modules (that includes the PIM-SM module). Similarly, the MFEA informs PIM-SM about any virtual interfaces information changes (*e.g.,* an address has been added or deleted) [5].

[5] Strictly speaking, the VIF (Virtual InterFace) abstraction is based on information that can be obtained directly from the kernel. However, the VIF information is also related with the RIB information as well, therefore for consistency reason, probably it is better if any VIF information changes is propagated via the RIB module to the PIM-SM module, especially because any change in the VIF information may affect the unicast routing as well. This is an open issue, therefore in the future the current behavior may be changed.

2.8 PimConfig Description

PimConfig handles the PIM-specific configuration [6]. This configuration is used to configure the following units:

- PimVif: protocol version, Hello-related options and timer values, etc.
- PimScopeZone table: add information about scoped zones.
- PimBsr: configure the local routing unit as a Candidate-BSR or a Candidate-RP.
- RpTable: add static RPs to the RP-set.

A Modification History

- December 11, 2002: Version 0.1 completed.

References

[1] XORP Multicast Routing Design Architecture. XORP technical document. http://www.xorp.org/.

[2] Bill Fenner, Mark Handley, Hugh Holbrook, and Isidor Kouvelas. Protocol Independent Multicast - Sparse Mode (PIM-SM): Protocol Specification (Revised). *Internet Draft, draft-ietf-pim-sm-v2-new-05.txt*, March 2002. Work in progress.

[3] Bill Fenner, Mark Handley, Roger Kermode, and David Thaler. Bootstrap Router (BSR) Mechanism for PIM Sparse Mode. *Internet Draft, draft-ietf-pim-sm-bsr-02.txt*, November 2001. Work in progress.

[6] Currently (December 2002), PimConfig is not implemented; rather, all state is kept into PimNode instead.