XORP Routing Information Base (RIB) Process
Version 0.4

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

August 28, 2003

1 Introduction

This document provides an overview of the XORP Routing Information Base (RIB) process. It is intended to provide a starting point for software developers wishing to understand or modify this software.

The RIB process takes routing information from multiple routing protocols, stores these routes, and decides which routes should be propagated on to the forwarding engine. The RIB performs the following tasks:

- Stores routes provided by the routing protocols running on a XORP router.
- If more than one routing protocol provides a route for the same subnet, the RIB decides which route will be used.
- The winning unicast routes are propagated to the Forwarding Engine Abstraction (FEA) process and hence on to the forwarding engine. Multicast routes are not propagated to the FEA - they are only used to provide topology information to multicast routing protocols.
- Protocols such as BGP may supply to the RIB routes that have a nexthop that is not an immediate neighbor. Such nexthops are resolved by the RIB so as to provide a route with an immediate neighbor to the FEA.
- Protocols such as BGP need to know routing metric and reachability information to nexthops that are not immediate neighbors. The RIB provides a way to register interest in such routing information, in such a way that the routing protocol will be notified if a change occurs.

By default, the RIB process holds four separate RIBs:

- Unicast IPv4 RIB
- Unicast IPv6 RIB
- Multicast IPv4 RIB

- Multicast IPv6 RIB

C++ templates are used to build specialized IPv4 and IPv6 versions from the same code. Routing protocols such as Multiprotocol BGP are capable of supplying routes that are multicast-only, and these would be stored in the multicast RIBs. The unicast and multicast RIBs primarily differ in that only unicast routes are propagated to the forwarding engine.

Note that we do not currently support multiple RIBs for other purposes, such as VPN support, but the RIB architecture will permit such extensions.

2 Structure of a RIB

The RIB process may hold multiple RIBs. Figure 1 gives an overview of the structure of a unicast RIB. In general, routing protocols supply routes to the OriginTables. These routes then flow though the tree

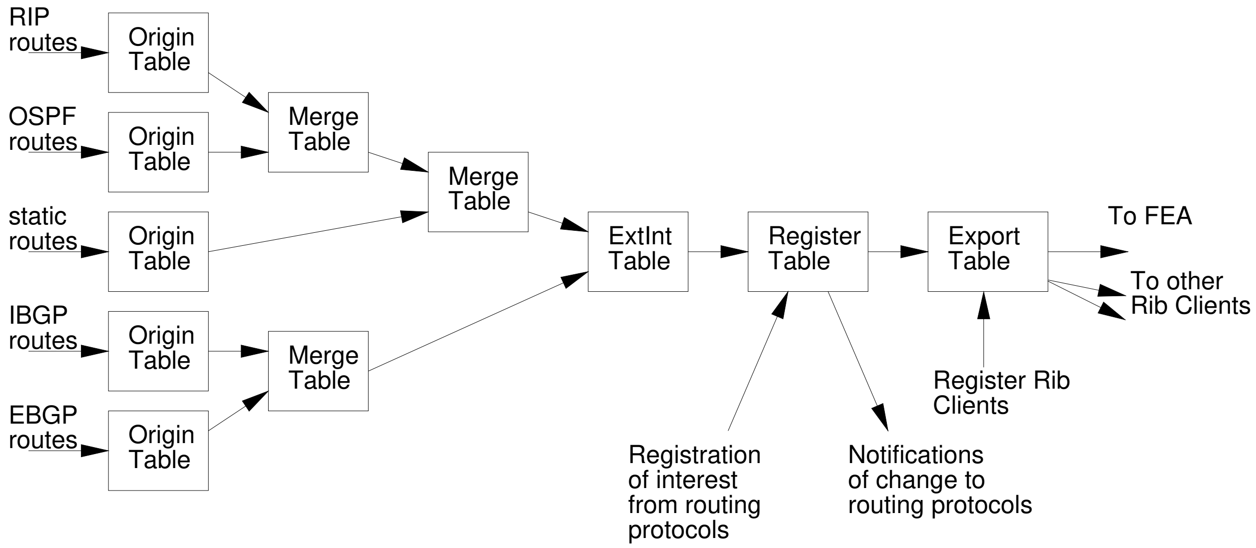

Figure 1: Overview of a RIB

structure from left to right, until they reach the export table, where they are propagated to the Forwarding Engine Abstraction (FEA) process.

2.1 Plumbing

The RIB plumbing code dynamically creates and maintains the tree of tables as shown in figure 1. When a new routing protocol registers with the RIB, a new OriginTable will be created, and a MergeTable or an ExtIntTable will be created to plumb that OriginTable into the RIB tree at an appropriate location. Similarly, if a routing protocol deregisters with the RIB, the relevant OriginTable and additional tables will be deleted and the RIB tree simplified again.

2.2 OriginTable

The OriginTable accepts `route_add` requests from a single routing protocol, stores the route, and propagates it downstream. It also answers `lookup_route` requests from downstream from the routes it has stored.

An OriginTable for the "connected" protocol always exists, to handle routes for directly connected interfaces. It gets its information via the VifManager from the Forwarding Engine Abstraction (FEA) process.

2.3 MergeTable

A MergeTable has two upstream (parent) tables and one downstream (child) table.

Multiple OriginTables may hold different routes for the same subnet. Thus, when an `add_route` request reaches a MergeTable from one parent, the MergeTable performs a route lookup on the other parent to see if the route already exists. If it does, then the new route is only propagated if it is better than the existing route, where "better" is determined based on the relative administrative distance of the routes.

Similarly, if a `delete_route` request reaches a MergeTable, it performs a route lookup on the other parent table. If the route being deleted was better than the alternative, then the delete is propagated downstream followed by an `add_route` for the alternative. If the route being deleted was worse than the alternative, then the deletion needs to be propagated no further.

When a MergeTable receives a `lookup_route` request from downstream, it sends the request on to both parents. The better of the two answers is sent in response.

2.4 ExtIntTable

ExtIntTable functions very similarly to the MergeTable, but there is an asymmetry between the parents. On the Internal side, the originating routing protocols always supply routes that have an immediate neighbor as the nexthop. On the External side, the originating routing protocols may supply routes that have an immediate neighbor as the nexthop, but they may also supply routes where the nexthop is multiple IP hops away.

When an `add_route` request arrives from the external parent, the ExtIntTable does the same comparisons that happen with a MergeTable. However, it also checks to see if the nexthop is an immediate neighbor. If it is not, then the ExtIntTable attempts to find a route that indicates which immediate neighbor to use to reach the nexthop, and the nexthop in the route that is propagated downstream will be that of this immediate neighbor. If no route exists to the nexthop, the route will not be propagated downstream, but will be stored in a table of unresolved routes in case a route that arrives later can cause it to resolve.

Each RIB only contains a single ExtIntTable.

2.5 RegisterTable

RegisterTable takes registrations from routing protocols for routing information related to specific destinations, answers the request, and stores the registration. If the routing information in the answer changes, it will asynchronously notify the routing protocol of the change. The precise interface is described in section 3.6, but the general idea is illustrated by this example:

Suppose the RIB contains routes for 1.0.0.0/16 and 1.0.2.0/24. Now a routing protocol expresses an interest in address 1.0.1.1. This matches against the entry for 1.0.0.0/16, so the answer contains 1.0.0.0/16 and the related nexthop and metric.

However we would like the routing protocol to be able to use the answer if it also cares about other addresses, such as 1.0.0.1 or 1.0.2.1. However, while the former matches against 1.0.0.0/16, the latter matches against 1.0.2.0/24. Thus if the RIB only returns 1.0.0.0/16, the routing protocol cannot tell whether it can use this information for any other address than the one it asked about.

To rectify this, the RIB returns not only the answer (1.0.0.0/16, plus metric and nexthop), but also the subset of this prefix for which the answer is known to be good. In this case, the answer is good for 1.0.0.0 to 1.0.1.255, which is returned as a subnet: 1.0.0.0/23. Any address that the routing protocol cares about that falls within 1.0.0.0/23 does not require additional communication with the RIB.

The RegisterTable keeps track of which information it gave which routing protocol, so that if this information becomes invalid, the routing protocol can be informed. For example, if a new route appears for 1.0.1.0/24, then this would cause the registration covering 1.0.0.0/23 to be invalidated because 1.0.1.0/24 is more specific than 1.0.0.0/16 and overlaps the range 1.0.0.0/23 of the registration.

Each RIB only contains a single RegisterTable.

2.6 ExportTable

The ExportTable exists to communicate to the FEA (and other registered processes) the changes that make it through to the end of the tree of tables. Both the unicast and multicast RIBs have an ExportTable.

By default the unicast RIBs export routes to the FEA, and the multicast RIBs don't export routes to anyone. Processes which need to see the whole routing table can register to be added to the export list for one or more RIBs. The ExportTable maintains this list.

Routes for directly connected subnets (from the "connected" OriginTable) are not propagated back to the FEA, as it already knows about these.

2.7 RedistTable

Note, August 2003: RedistTable is unfinished.

A RedistTable can by dynamically plumbed into the tree of tables at any point, but it is most likely to be inserted immediately after an OriginTable. The purpose of a RedistTable is to allow the configured (and possibly filtered) redistribution of routes from one routing protocol to another. For example, routes from within an AS might be propagated from OSPF to BGP for external advertisement.

There may be many RedistTables in each RIB.

3 XRL Interface

The RIB supports the following XRLs.

3.1 Routing Protocol Registration

```
add_igp_table4 ? protocol:txt & unicast:bool & multicast:bool
add_igp_table6 ? protocol:txt & unicast:bool & multicast:bool
add_egp_table4 ? protocol:txt & unicast:bool & multicast:bool
add_egp_table6 ? protocol:txt & unicast:bool & multicast:bool
delete_igp_table4 ? protocol:txt & unicast:bool & multicast:bool
delete_igp_table6 ? protocol:txt & unicast:bool & multicast:bool
```

```
delete_egp_table4 ? protocol:txt & unicast:bool & multicast:bool
delete_egp_table6 ? protocol:txt & unicast:bool & multicast:bool
```

These XRLs are used by routing protocols to register with the RIB process, and hence to create OriginTables in all the RIBs.

The `_igp_` xrls will plumb the OriginTable on the Internal side of the ExtIntTable. The `_egp_` xrls will plumb the OriginTable on the External side of the ExtIntTable.

`protocol` is a text string used to identify the routing protocol. Currently it MUST be a protocol the RIB knows about, or an administrative distance cannot be assigned to the routes. Future versions of the RIB may make this interface more extensible.

`unicast` and `multicast` indicate whether the routing protocol will insert routes into the unicast RIB or the multicast RIB or both.

3.2 Adding and Deleting Routes

```
add_route4 ? protocol:txt & unicast:bool & multicast:bool &
        network:ipv4net & nexthop:ipv4 & metric:u32

add_route6 ? protocol:txt & unicast:bool & multicast:bool &
        network:ipv6net & nexthop:ipv6 & metric:u32

replace_route4 ? protocol:txt & unicast:bool & multicast:bool &
        network:ipv4net & nexthop:ipv4 & metric:u32

replace_route6 ? protocol:txt & unicast:bool & multicast:bool &
        network:ipv6net & nexthop:ipv6 & metric:u32

delete_route4 ? protocol:txt & unicast:bool & multicast:bool &
        network:ipv4net

delete_route6 ? protocol:txt & unicast:bool & multicast:bool &
        network:ipv6net
```

These XRLs are used to communicate new routes, changed routes, or the deletion of routes to the RIB.

Note that sending an `add_route` for a route that is already in the OriginTable for that protocol is an error, as is sending a `replace_route` or `delete_route` for a route that is not in the OriginTable for that protocol.

3.3 Route Lookup

```
lookup_route4 ? addr:ipv4 & unicast:bool & multicast:bool
        -> nexthop:ipv4
lookup_route6 ? addr:ipv6 & unicast:bool & multicast:bool
        -> nexthop:ipv6
```

These XRLs may be used to see how the RIB would route a packet for a specific destination.

nexthop will return the resolved nexthop if the request is successful, or all zeros otherwise. It is an error for the unicast and multicast fields to both be true or both false.

3.4 VIF Management (test interface)

```
new_vif ? name:txt
add_vif_addr4 ? name:txt & addr:ipv4 & subnet:ipv4net
add_vif_addr6 ? name:txt & addr:ipv6 & subnet:ipv6net
```

These XRLs can be used to inform the RIB about directly connected (virtual) interfaces. The use of these XRLs is intended only for testing purposes - the RIB normally learns of VIFs directly from the FEA process.

3.5 Route Redistribution

```
redist_enable4 ? from:txt & to:txt & unicast:bool & multicast:bool
redist_enable6  ? from:txt & to:txt & unicast:bool & multicast:bool
redist_disable4 ? from:txt & to:txt & unicast:bool & multicast:bool
redist_disable6 ? from:txt & to:txt & unicast:bool & multicast:bool
```

These XRLs are intended to be used to enable route redistribution between routing protocols. However, this interface is currently (August 2003) incomplete and should not be used.

3.6 Registration of Interest in Routes

```
register_interest4 ? target:txt & addr:ipv4
        -> resolves:bool & base_addr:ipv4 & prefix:u32 &
           realprefix:u32 & nexthop:ipv4 & metric:u32

deregister_interest4 ?  target:txt & addr:ipv4 & prefix:u32

register_interest6 ? target:txt & addr:ipv6
        -> resolves:bool & base_addr:ipv6 & prefix:u32 &
           realprefix:u32 & nexthop:ipv6 & metric:u32

deregister_interest6 ?  target:txt & addr:ipv6 & prefix:u32
```

These XRLs are used to register and deregister interest in routing information related to a specific IP address.

Target is the name of the XRL module that registered the interest.

resolves indicates whether or not the address is routable. If it is not routable, the values of nexthop and metric are undefined. realprefix returns the prefix length of the routing entry that matches the address in the request. prefix returns the prefix length of the largest subnet that covers the address and is not overlayed by a more specific route. Thus prefix <= realprefix and addr is an address within the subnet base_addr/prefix which itself is a subset of the subnet base_addr/realprefix.

The routing protocol need not ask again for any address that lies within base_addr/prefix but can not use this answer to determine anything about addresses that lie outside of base_addr/prefix but within base_addr/realprefix.

3.7 Registration of Interest in Routes: Client Interface

When a routing protocol has registered interest in routes, the RIB will need to be able to asynchronously call the routing protocol to inform it of any changes. Thus the routing protocol must implement the rib client interface. This consists of the following XRLs:

```
route_info_changed4 ? addr:ipv4 & prefix_len:u32 &
        nexthop:ipv4 & metric:u32
route_info_changed6 ? addr:ipv6 & prefix_len:u32 &
        nexthop:ipv6 & metric:u32

route_info_invalid4 ? addr:ipv4 & prefix_len:u32
route_info_invalid6 ? addr:ipv6 & prefix_len:u32
```

The `route_info_changed` XRLs inform the routing protocol that the nexthop or metric associated with the route with subnet `addr/prefix_len` has changed. The registration with the RIB is still valid.

The `route_info_invalid` XRLs inform the routing protocol that the information associated with the specified subnet is no longer correct. The registration with the RIB is no longer valid, and the routing protocol must re-register with the RIB to find out what happened.

3.8 Adding and deleting RIB clients

```
add_rib_client4 ? target_name:txt & unicast:bool & multicast:bool
add_rib_client6 ? target_name:txt & unicast:bool & multicast:bool

delete_rib_client4 ? target_name:txt & unicast:bool & multicast:bool
delete_rib_client6 ? target_name:txt & unicast:bool & multicast:bool

enable_rib_client4 ? target_name:txt & unicast:bool & multicast:bool
enable_rib_client6 ? target_name:txt & unicast:bool & multicast:bool

disable_rib_client4 ? target_name:txt & unicast:bool & multicast:bool
disable_rib_client6 ? target_name:txt & unicast:bool & multicast:bool
```

The RIB's ExportTable is responsible for exporting all the winning unicast routes and any changes of those routes to the FEA. However from the RIB's point of view the FEA is just a RibClient, and other processes may also register to be RibClients and to receive all the routing information. By default the FEA is a RibClient for unicast routes only (it doesn't need to register). Processes may use the `add_rib_client` XRLs above to become a RibClient for unicast routes, for multicast routes, or both. They may also enable and disable the propagation of this routing information, or deregister completely using `delete_rib_client`.