Network Working Group
Internet-Draft
Intended status: Informational
Expires: February 22, 2014

N. Bahadur, Ed.
R. Folkes, Ed.
Juniper Networks, Inc.
S. Kini
Ericsson
J. Medved
Cisco

August 21, 2013

Routing Information Base Info Model
draft-nitinb-i2rs-rib-info-model-02

Abstract

Routing and routing functions in enterprise and carrier networks are typically performed by network devices (routers and switches) using a routing information base (RIB). Protocols and configuration push data into the RIB and the RIB manager install state into the hardware; for packet forwarding. This draft specifies an information model for the RIB to enable defining a standardized data model. Such a data model can be used to define an interface to the RIB from an entity that may even be external to the network device. This interface can be used to support new use-cases being defined by the IETF I2RS WG.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on February 22, 2014.

Table of Contents

1. Introduction

Routing and routing functions in enterprise and carrier networks are traditionally performed in network devices. Traditionally routers run routing protocols and the routing protocols (along with static config) populates the Routing information base (RIB) of the router. The RIB is managed by the RIB manager and it provides a north-bound interface to its clients i.e. the routing protocols to insert routes into the RIB. The RIB manager consults the RIB and decides how to program the forwarding information base (FIB) of the hardware by interfacing with the FIB-manager. The relationship between these entities is shown in Figure 1.

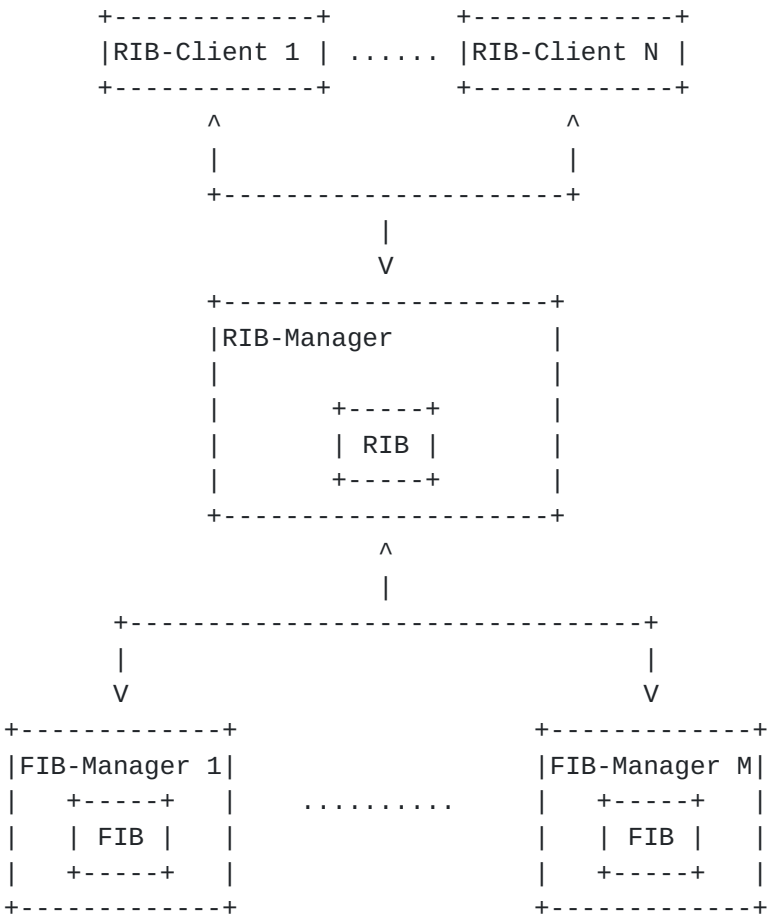


Figure 1: RIB-Manager, RIB-Clients and FIB-Managers

Routing protocols are inherently distributed in nature and each router makes an independent decision based on the routing data received from its peers. With the advent of newer deployment paradigms and the need for specialized applications, there is an emerging need to guide the router's routing function [I-D.atlas-i2rs-problem-statement]. Traditional network-device

protocol-based RIB population suffices for most use cases where distributed network control works. However there are use cases in which the network admins today configure static routes, policies and RIB import/export rules on the routers. There is also a growing list of use cases [I-D.white-i2rs-use-case], [I-D.hares-i2rs-use-case-vn-vc] in which a network admin might want to program the RIB based on data unrelated to just routing (within that network's domain). It could be based on routing data in adjacent domain or it could be based on load on storage and compute in the given domain. Or it could simply be a programmatic way of creating on-demand dynamic overlays between compute hosts (without requiring the hosts to run traditional routing protocols). If there was a standardized programmatic interface to a RIB, it would fuel further networking applications targeted towards specific niches.

A programmatic interface to the RIB involves 2 types of operations - reading what's in the RIB and adding/modifying/deleting contents of the RIB. [I-D.white-i2rs-use-case] lists various use-cases which require read and/or write manipulation of the RIB.

In order to understand what is in a router's RIB, methods like per-protocol SNMP MIBs and show output screen scraping are being used. These methods are not scalable, since they are client pull mechanisms and not proactive push (from the router) mechanisms. Screen scraping is error prone (since the output format can change) and vendor dependent. Building a RIB from per-protocol MIBs is error prone since the MIB data represents protocol data and not the exact information that went into the RIB. Thus, just getting read-only RIB information from a router is a hard task.

Adding content to the RIB from an external entity can be done today using static configuration support provided by router vendors. However the mix of what can be modified in the RIB varies from vendor to vendor and the way of configuring it is also vendor dependent. This makes it hard for an external entity to program a multi-vendor network in a consistent and vendor independent way.

The purpose of this draft is to specify an information model for the RIB. Using the information model, one can build a detailed data model for the RIB. And that data model could then be used by an external entity to program a network device.

The rest of this document is organized as follows. Section 2 goes into the details of what constitutes and can be programmed in a RIB. Guidelines for reading and writing the RIB are provided in Section 3 and Section 4 respectively. Section 5 provides a high-level view of the events and notifications going from a network device to an external entity, to update the external entity on asynchronous

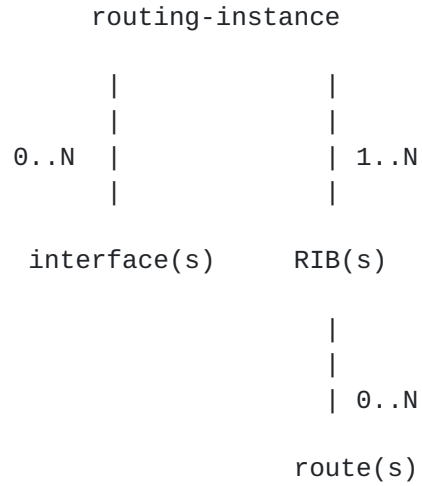
events. The RIB grammar is specified in Section 6. Examples of using the RIB grammar are shown in Section 7. Section 8 covers considerations for performing RIB operations at scale.

1.1. Conventions used in this document

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

2. RIB data

This section describes the details of a RIB. It makes forward references to objects in the RIB grammar (Section 6). A high-level description of the RIB contents is as shown below.



2.1. RIB definition

A RIB is an entity that contains routes. A RIB is identified by its name and a RIB is contained within a routing instance (Section 2.2). The name MUST be unique within a routing instance. All routes in a given RIB MUST be of the same type (e.g. IPv4). Each RIB MUST belong to some routing instance.

A RIB can be tagged with a MULTI_TOPOLOGY_ID. If a routing instance is divided into multiple logical topologies, then the multi-topology field is used to distinguish one topology from the other, so as to keep routes from one topology independent of routes from another topology.

If a routing instance contains multiple RIBs of the same type (e.g. IPv4), then a MULTI_TOPOLOGY_ID MUST be associated with each such

RIB. Multiple RIBs are useful when describing multiple topology IGP (Interior Gateway Protocol) networks (see [RFC4915] and [RFC5120]). In a given routing instance, MULTI_TOPOLOGY_ID MUST be unique across RIBs of the same type.

Each RIB can be optionally associated with a ENABLE_IP_RPF_CHECK attribute that enables Reverse path forwarding (RPF) checks on all IP routes in that RIB. Reverse path forwarding (RPF) check is used to prevent spoofing and limit malicious traffic. For IP packets, the IP source address is looked up and the rpf interface(s) associated with the route for that IP source address is found. If the incoming IP packet's interface matches one of the rpf interface(s), then the IP packet is forwarded based on its IP destination address; otherwise, the IP packet is discarded.

2.2. Routing instance

A routing instance, in the context of the RIB information model, is a collection of RIBs, interfaces, and routing parameters. A routing instance creates a logical slice of the router and allows different logical slices; across a set of routers; to communicate with other each. Layer 3 Virtual Private Networks (VPN), Layer 2 VPNs (L2VPN) and Virtual Private Lan Service (VPLS) can be modeled as routing instances. Note that modeling a Layer 2 VPN using a routing instance only models the Layer-3 (RIB) aspect and does not model any layer-2 information (like ARP) that might be associated with the L2VPN.

The set of interfaces indicates which interfaces are associated with this routing instance. The RIBs specify how incoming traffic is to be forwarded. And the routing parameters control the information in the RIBs. The intersection set of interfaces of 2 routing instances SHOULD be the null set. In other words, an interface MUST NOT be present in 2 routing instances. Thus a routing instance describes the routing information and parameters across a set of interfaces.

A routing instance MUST contain the following mandatory fields.

- INSTANCE_NAME: A routing instance is identified by its name, INSTANCE_NAME. This SHOULD be unique across all routing instances in a given network device.
- INSTANCE_DISTINGUISHER: Each routing instance MUST have a distinguisher associated with it. It enables one to distinguish routes across routing instances. The route distinguisher MUST be unique across all routing instances in a given network device. How the INSTANCE_DISTINGUISHER is allocated and kept unique is outside the scope of this document. The instance distinguisher maps well to BGP route-distinguisher for virtual private networks (VPNs). However, the same concept can be used for other use-cases as well.

- rib-list: This is the list of RIBs associated with this routing instance. Each routing instance can have multiple RIBs to represent routes of different types. For example, one would put IPv4 routes in one RIB and MPLS routes in another RIB.

A routing instance MAY contain the following optional fields.

- interface-list: This represents the list of interfaces associated with this routing instance. The interface list helps constrain the boundaries of packet forwarding. Packets coming on these interfaces are directly associated with the given routing instance. The interface list contains a list of identifiers, with each identifier uniquely identifying an interface.
- ROUTER_ID: The router-id field identifies the network device in control plane interactions with other network devices. This field is to be used if one wants to virtualize a physical router into multiple virtual routers. Each virtual router MUST have a unique router-id. ROUTER_ID MUST be unique across all network devices in a given domain.
- as-data: This is an identifier of the administrative domain to which the routing instance belongs. The as-data fields is used when the routes in this instance are to be tagged with certain autonomous system (AS) characteristics. The RIB manager can use AS length as one of the parameters for making route selection. as-data consists of a AS number and an optional Confederation AS number ([RFC5065]).

2.3. Route

A route is essentially a match condition and an action following the match. The match condition specifies the kind of route (IPv4, MPLS, etc.) and the set of fields to match on. Figure 2 represents the overall contents of a route.

artwork

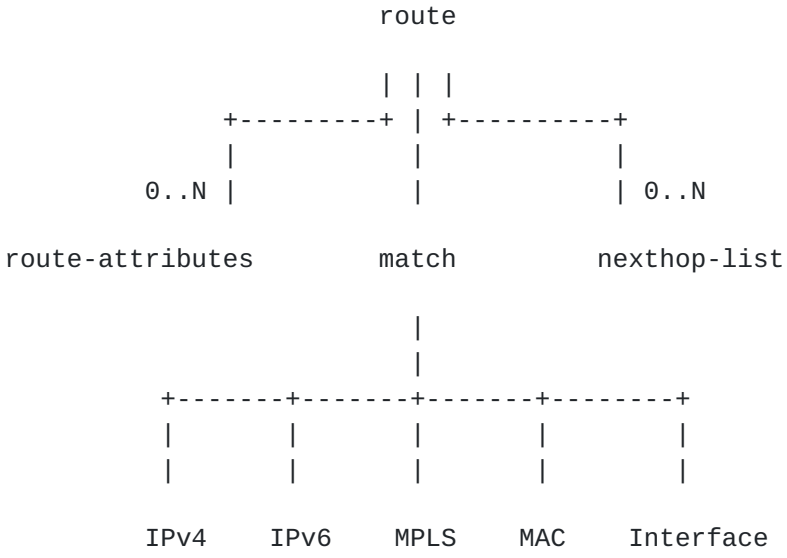


Figure 2: Route model

This document specifies the following match types:

- IPv4: Match on destination IP in IPv4 header
- IPv6: Match on destination IP in IPv6 header
- MPLS: Match on a MPLS tag
- MAC: Match on ethernet destination addresses
- Interface: Match on incoming interface of packet
- IP multicast: Match on (S, G) or (*, G), where S and G are IP prefixes

Each route can have associated with it one or more optional route attributes.

- ROUTE_PREFERENCE: This is a numerical value that allows for comparing routes from different protocols (where static configuration is also considered a protocol for the purpose of this field). It is also known as administrative-distance. The lower the value, the higher the preference. For example there can be an OSPF route for 192.0.2.1/32 with a preference of 5. If a controller programs a route for 192.0.2.1/32 with a preference of 2, then the controller entered route will be preferred by the RIB manager. Preference should be used to dictate behavior. For more examples of preference, see Section 7.1.
- ROUTE_METRIC: Route preference is used for comparing routes from different protocols. Route metric is used for comparing routes learned by the same protocol. If a controller wishes to program 2 or more routes to the same destination, then it can use the metric field to disambiguate the 2 routes. For more examples, see Section 7.1.

- LOCAL_ONLY: This is a boolean value. If this is present, then it means that this route should not be exported into other RIBs or other RIBs.
- rpf-check-interface: Reverse path forwarding (RPF) check is used to prevent spoofing and limit malicious traffic. For IP packets, the IP source address is looked up and the rpf-check-interface associated with the route for that IP source address is found. If the incoming IP packet's interface matches one of the rpf-check-interfaces, then the IP packet is forwarded based on its IP destination address; otherwise, the IP packet is discarded. For MPLS routes, there is no source address to be looked up, so the usage is slightly different. For an MPLS route, a packet with the specified MPLS label will only be forwarded if it is received on one of the interfaces specified by the rpf-check-interface. If no rpf-check-interface is specified, then matching packets are no subject to this check. This field overrides the ENABLE_IP_RPF_CHECK flag on the RIB and interfaces provided in this list are used for doing the RPF check.
- as-path: A route can have an as-path associated with it to indicate which set of autonomous systems has to be traversed to reach the final destination. The as-path attribute can be used by the RIB manager in multiple ways. The RIB manager can choose paths with lower as-path length. Or the RIB manager can choose to not install paths going via a particular AS. How exactly the RIB manager uses the as-path is outside the scope of this document. For details of how the as-path is formed, see Section 5.1.2 of [RFC4271] and Section 3 of [RFC5065].
- route-vendor-attributes: Vendors can specify vendor-specific attributes using this. The details of this field is outside the scope of this document.

2.4. Nexthop

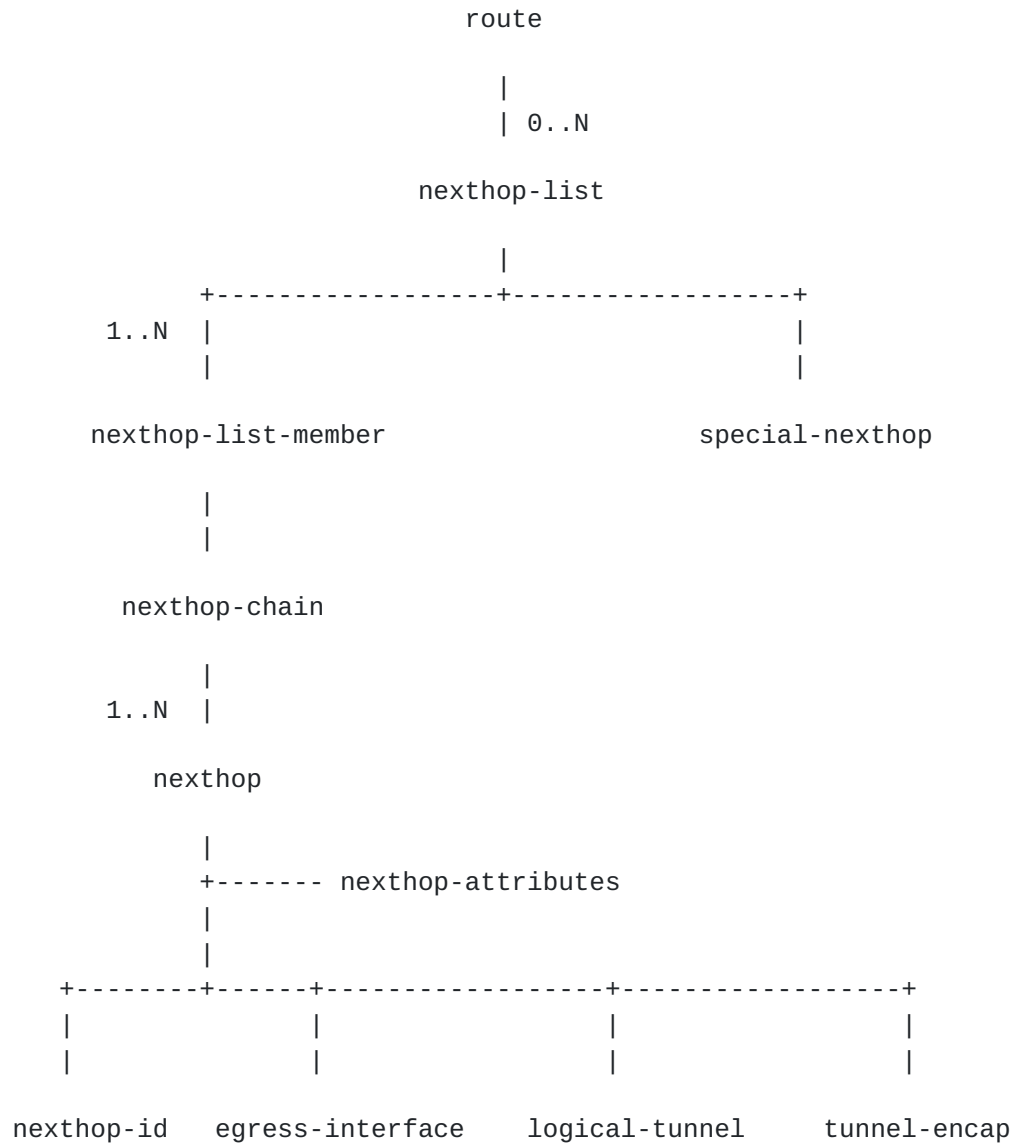
A nexthop represents an object or action resulting from a route lookup. For example, if a route lookup results in sending the packet out a given interface, then the nexthop represents that interface.

Nexthops can be fully resolved nexthops or unresolved nexthop. A resolved nexthop is something that is ready for installation in the FIB. For example, a nexthop that points to an interface. An unresolved nexthop is something that requires the RIB manager to figure out the final resolved nexthop. For example, a nexthop could point to an IP address. The RIB manager has to resolve how to reach that IP address - is the IP address reachable by regular IP forwarding or by a MPLS tunnel or by both. If the RIB manager cannot resolve the nexthop, then the nexthop stays in unresolved state and is NOT a candidate for installation in the FIB. Future RIB events can cause a nexthop to get resolved (like that IP address being

advertised by an IGP neighbor).

The RIB information model allows an external entity to program nexthops that may be unresolved initially. Whenever a unresolved nexthop gets resolved, the RIB manager will send a notification of the same (see Section 5).

The overall structure and usage of a nexthop is as shown in the figure below.



Nexthops can be identified by an identifier to create a level of indirection. The identifier is set by the RIB manager and returned to the external entity on request. The RIB data-model SHOULD support a way to optionally receive a nexthop identifier for a given nexthop.

For example, one can create a nexthop that points to a BGP peer. The returned nexthop identifier can then be used for programming routes to point to the same nexthop. Given that the RIB manager has created an indirection for that BGP peer using the nexthop identifier, if the transport path to the BGP peer changes, that change in path will be seamless to the external entity and all routes that point to that BGP peer will automatically start going over the new transport path. Nexthop indirection using identifier could be applied to not just unicast nexthops, but even to nexthops that contain chains and nested nexthops (Section 2.4.1).

2.4.1. Nexthop types

This document specifies a very generic, extensible and recursive grammar for nexthops. Nexthops can be

- Unicast nexthops - pointing to an interface
- Tunnel nexthops - pointing to a tunnel
- Replication lists - list of nexthops to which to replicate a packet to
- Weighted lists - for load-balancing
- Protection lists - for primary/backup paths
- Nexthop chains - for chaining headers, e.g. MPLS label over a GRE header
- Lists of lists - recursive application of the above
- Indirect nexthops - pointing to a nexthop identifier
- Special nexthops - for performing specific well-defined functions

It is expected that all network devices will have a limit on how many levels of lookup can be performed and not all hardware will be able to support all kinds of nexthops. RIB capability negotiation becomes very important for this reason and a RIB data-model MUST specify a way for an external entity to learn about the network device's capabilities. Examples of when and how to use various kinds of nexthops are shown in Section 7.2.

Tunnel nexthops allow an external entity to program static tunnel headers. There can be cases where the remote tunnel end-point does not support dynamic signaling (e.g. no LDP support on a host) and in those cases the external entity might want to program the tunnel header on both ends of the tunnel. The tunnel nexthop is kept generic with specifications provided for some commonly used tunnels. It is expected that the data-model will model these tunnel types with complete accuracy.

Nexthop chains can be used to specify multiple headers over a packet, before a packet is forwarded. One simple example is that of MPLS over GRE, wherein the packet has a inner MPLS header followed by a GRE header followed by an IP header. The outermost IP header is decided by the network device whereas the MPLS header and GRE header

are specified by the controller. Not every network device will be able to support all kinds of nexthop chains and an arbitrary number of header chained together. The RIB data-model SHOULD provide a way to expose nexthop chaining capability supported by a given network device.

2.4.2. Nexthop list attributes

For nexthops that are of the form of a list(s), attributes can be associated with each member of the list to indicate the role of an individual member of the list. Two kinds of attributes are specified:

- PROTECTION_PREFERENCE: This provides a primary/backup like preference. The preference is an integer value that should be set to 1 or 2. Nexthop members with a preference of 1 are preferred over those with preference of 2. The network device SHOULD create a list of nexthops with preference 1 (primary) and another list of nexthops with preference 2 (backup) and SHOULD pre-program the forwarding plane with both the lists. In case if all the primary nexthops fail, then traffic MUST be switched over to members of the backup nexthop list. All members in a list MUST either have a protection preference specified or all members in a list MUST NOT have a protection preference specified.
- LOAD_BALANCE_WEIGHT: This is used for load-balancing. Each list member MUST be assigned a weight. The weight is a percentage number from 1 to 99. The weight determines how much traffic is sent over a given list member. If one of the members nexthops in the list is not active, then the weight value of that nexthop SHOULD be distributed among the other active members. How the distribution is done is up to the network device and not in the scope of the document. In other words, traffic should always be load-balanced even if there is a failure. After a failure, the external entity SHOULD re-program the nexthop list with updated weights so as to get a deterministic behavior among the remaining list members. To perform equal load-balancing, one MAY specify a weight of "0" for all the member nexthops. The value "0" is reserved for equal load-balancing and if applied, MUST be applied to all member nexthops.

A nexthop list MAY contain elements that have both PROTECTION_PREFERENCE and LOAD_BALANCE_WEIGHT set. When both are set, it means under normal operation the network device should load balance the traffic over all nexthops with a protection preference of 1. And when all nexthops with a protection preference of 1 are down (or unavailable), then traffic MUST be load balanced over elements with protection preference of 2.

2.4.3. Nexthop content

At the lowest level, a nexthop can point to a:

- identifier: This is an identifier returned by the network device representing another nexthop or another nexthop chain.
- EGRESS_INTERFACE: This represents a physical, logical or virtual interface on the network device.
- address: This can be an IP address or MAC address or ISO address.
  - An optional RIB name can also be specified to indicate the RIB in which the address is to be looked up further. One can use the RIB name field to direct the packet from one domain into another domain. For example, a MPLS packet coming in on an interface would be looked up in a MPLS RIB and the nexthop for that could indicate that we strip the MPLS label and do a subsequent IPv4 lookup in an IPv4 RIB. By default the RIB will be the same in which the route lookup was performed.
  - An optional egress interface can be specified to indicate which interface to send the packet out on. The egress interface is useful when the network device contains Ethernet interfaces and one needs to perform an ARP lookup for the IP packet.
- tunnel encap: This can be an encap representing an IP tunnel or MPLS tunnel or others as defined in this document. An optional egress interface can be specified to indicate which interface to send the packet out on. The egress interface is useful when the network device contains Ethernet interfaces and one needs to perform an ARP lookup for the IP packet.
- logical tunnel: This can be a MPLS LSP or a GRE tunnel (or others as defined in this document), that is represented by a unique identifier (E.g. name).
- RIB_NAME: A nexthop pointing to a RIB indicates that the route lookup needs to continue in the specified RIB. This is a way to perform chained lookups.

2.4.4. Nexthop attributes

Certain information is encoded implicitly in the nexthop and does not need to be specified by the controller. For example, when a IP packet is forwarded out, the IP TTL is decremented by default. Same applies for an MPLS packet. Similarly, when an IP packet is sent over an ethernet interface, any ARP processing is handled implicitly by the network device and does not need to be programmed by an external device.

A nexthop can have some attributes associated with it. The purpose of the attributes is to either override implicit behavior (like that related to TTL processing) or to guide the network device to perform something specific. Vendor specific attributes can also be specified. The details of vendor specific attributes is outside the

scope of this document.

2.4.4.1. Nexthop flags

Nexthop flags in a nexthop is an optional attribute that is used to denote specific connotation to hardware. Two common types of operations are specified using nexthop flags.

- NO_DECREMENT_TTL: This indicates that the IPv4 time-to-live field in an IPv4 packet MUST NOT be decremented before the packet is forwarded. This may be applied one when an IPv4 packet is encapsulated in a tunnel (E.g. MPLS) and one wants to hide the fact that the packet is going through a tunnel.
- NO_PROPAGATE_TTL: This indicates that the IPv4 time-to-live field in an IPv4 packet MUST NOT be propagated into an equivalent field, when the IPv4 packet is tunneled. For example, if the IPv4 packet is tunneled over MPLS, then the network device should use the default time-to-live value for the outer MPLS header. This field can also be used to indicate that when a tunnel terminates, one does not propagate the outer header's time-to-live value into the inner header. So, on MPLS tunnel termination, one does not propagate the MPLS TTL value into the IPv4 header.

The TTL nexthop flags can be used to simulate a Pipe model for tunnels. See [RFC3443] for a detailed understanding of Pipe model and Uniform model.

2.4.5. Nexthop vendor attributes

This field has been defined for vendor specific extensions. The contents of this field are beyond the scope of this document.

2.4.6. Special nexthops

This document specifies certain special nexthops. The purpose of each of them is explained below:

- DISCARD: This indicates that the network device should drop the packet and increment a drop counter.
- DISCARD_WITH_ERROR: This indicates that the network device should drop the packet, increment a drop counter and send back an appropriate error message (like ICMP error).
- RECEIVE: This indicates that that the traffic is destined for the network device. For example, protocol packets or OAM packets. All locally destined traffic SHOULD be throttled to avoid a denial of service attack on the router's control plane. An optional rate-limiter can be specified to indicate how to throttle traffic destined for the control plane. The description of the rate-limiter is outside the scope of this document.

3. Reading from the RIB

A RIB data-model MUST allow an external entity to read entries, for RIBs created by that entity. The network device administrator MAY allow reading of other RIBs by an external entity through access lists on the network device. The details of access lists are outside the scope of this document.

The data-model MUST support a full read of the RIB and subsequent incremental reads of changes to the RIB. An external agent SHOULD be able to request a full read at any time in the lifecycle of the connection. When sending data to an external entity, the RIB manager SHOULD try to send all dependencies of an object prior to sending that object.

4. Writing to the RIB

A RIB data-model MUST allow an external entity to write entries, for RIBs created by that entity. The network device administrator MAY allow writes to other RIBs by an external entity through access lists on the network device. The details of access lists are outside the scope of this document.

When writing an object to a RIB, the external entity SHOULD try to write all dependencies of the object prior to sending that object. The data-model MUST support requesting identifiers for nexthops and collecting the identifiers back in the response.

Route programming in the RIB MUST result in a return code that contains the following attributes:

- Installed - Yes/No (Indicates whether the route got installed in the FIB)
- Active - Yes/No (Indicates whether a route is fully resolved and is a candidate for selection)
- Reason - E.g. Not authorized

The data-model MUST specify which objects are modify-able objects. A modify-able object is one whose contents can be changed without having to change objects that depend on it and without affecting any data forwarding. To change a non-modifiable object, one will need to create a new object and delete the old one. For example, routes that use a nexthop that is identifier by a nexthop-identifier should be unaffected when the contents of that nexthop changes.

5. Events and Notifications

Asynchronous notifications are sent by the network device's RIB

manager to an external entity when some event occurs on the network device. A RIB data-model MUST support sending asynchronous notifications. A brief list of suggested notifications is as below:

- Route change notification, with return code as specified in Section 4
- Nexthop resolution status (resolved/unresolved) notification

6. RIB grammar

This section specifies the RIB information model in Routing Backus-Naur Form [RFC5511].

```
<routing-instance> ::= <INSTANCE_NAME> <INSTANCE_DISTINGUISHER>
                       [<interface-list>] <rib-list>
                       [<ROUTER_ID>] [<as-data>]

<as-data> ::= <AS_NUMBER> [<CONFEDERATION_AS>]

<interface-list> ::= (<INTERFACE_IDENTIFIER> ...)

<rib-list> ::= (<rib> ...)
<rib> ::= <RIB_NAME> <rib-family>
                    [<route> ... ] [<MULTI_TOPOLOGY_ID>]
                    [ENABLE_IP_RPF_CHECK]
<rib-family> ::= <IPV4_RIB_FAMILY> | <IPV6_RIB_FAMILY> |
                 <MPLS_RIB_FAMILY> | <IEEE_MAC_RIB_FAMILY>

<route> ::= <match> <nexthop-list>
            [<route-attributes>]
            [<route-vendor-attributes>]


<match> ::= <ipv4-route> | <ipv6-route> | <mpls-route> |
            <mac-route> | <interface-route>

<ipv4-route> ::= <ipv4-prefix> [<multicast-source-ipv4-address>]
<ipv4-prefix> ::= <IPV4_ADDRESS> <IPV4_ADDRESS_LENGTH>

<ipv6-route> ::= <ipv6-prefix> [<multicast-source-ipv6-address>]
<ipv6-prefix> ::= <IPV6_ADDRESS> <IPV6_PREFIX_LENGTH>

<mpls-route> ::= <MPLS> <MPLS_LABEL>
<mac-route> ::= <IEEE_MAC> ( <MAC_ADDRESS> )
<interface-route> ::= <INTERFACE> <INTERFACE_IDENTIFIER>

<multicast-source-ipv4-address> ::= <IPV4_ADDRESS>
                                    <IPV4_PREFIX_LENGTH>
```

```
   <multicast-source-ipv6-address> ::= <IPV6_ADDRESS>
                                       <IPV6_PREFIX_LENGTH>

   <route-attributes> ::= [<ROUTE_PREFERENCE>] [<ROUTE_METRIC>]
                          [<LOCAL_ONLY>]
                          [<address-family-route-attributes>]

   <address-family-route-attributes> ::= <ip-route-attributes> |
                                         <mpls-route-attributes> |
                                         <ethernet-route-attributes>
   <ip-route-attributes> ::= [<as-path>] [<rpf-check-interface>]
   <as-path> ::= (<as-path-segment-type> <as-list>) [<as-path> ...]
   <as-path-segment-type> ::= <AS_SET> | <AS_SEQUENCE> |
                              <AS_CONFED_SEQUENCE> | <AS_CONFED_SET>
   <as-list> ::= (<AS_NUMBER> ...) [<as-path>]


   <rpf-check-interface> ::= <interface-list>

   <mpls-route-attributes> ::= [<rpf-check-interface>]
   <ethernet-route-attributes> ::= <>
   <route-vendor-attributes> ::= <>

   <nexthop-list> ::= <special-nexthop> |
                      ((<nexthop-list-member>) |
                       ([<nexthop-list-member> ... ] <nexthop-list> ))

   <nexthop-list-member> ::= (<nexthop-chain> |
                              <nexthop-chain-identifier> )
                             [<nexthop-list-member-attributes>]
   <nexthop-list-member-attributes> ::= [<PROTECTION_PREFERENCE>]
                                        [<LOAD_BALANCE_WEIGHT>]

   <nexthop-chain> ::= (<nexthop> ...)
   <nexthop-chain-identifier> ::= <NEXTHOP_NAME> | <NEXTHOP_ID>
   <nexthop> ::= (<nexthop-identifier> | <EGRESS_INTERFACE> |
                  (<nexthop-address>
                     ([<RIB_NAME>] | [<EGRESS_INTERFACE>])) |
                  (<tunnel-encap> [<EGRESS_INTERFACE>]) |
                  <logical-tunnel> |
                  <RIB_NAME>)
                  [<nexthop-attributes>]
                  [<nexthop-vendor-attributes>]

   <nexthop-identifier> ::= <NEXTHOP_NAME> | <NEXTHOP_ID>
   <nexthop-address> ::= (<IPv4> <ipv4-address>) |
                         (<IPV6> <ipv6-address>) |
                         (<IEEE_MAC> <IEEE_MAC_ADDRESS>) |
```

```
                        (<ISO> <ISO_ADDRESS>)
   <special-nexthop> ::= <DISCARD> | <DISCARD_WITH_ERROR> |
                        (<RECEIVE> [<COS_VALUE>] [<rate-limiter>])
   <rate-limiter> ::= <>

   <logical-tunnel> ::= <tunnel-type> <TUNNEL_NAME>
   <tunnel-type> ::= <IP> | <MPLS> | <GRE> | <VxLAN> | <NVGRE>

   <tunnel-encap> ::= (<IPV4> <ipv4-header>) |
                      (<IPV6> <ipv6-header>) |
                      (<MPLS> <mpls-header>) |
                      (<GRE> <gre-header>) |
                      (<VXLAN> <vxlan-header>) |
                      (<NVGRE> <nvgre-header>)

   <ipv4-header> ::= <SOURCE_IPv4_ADDRESS> <DESTINATION_IPv4_ADDRESS>
                     <PROTOCOL> [<TTL>] [<DSCP>]

   <ipv6-header> ::= <SOURCE_IPV6_ADDRESS> <DESTINATION_IPV6_ADDRESS>
                     <NEXT_HEADER> [<TRAFFIC_CLASS>]
                     [<FLOW_LABEL>] [<HOP_LIMIT>]

   <mpls-header> ::= (<mpls-label-operation> ...)
   <mpls-label-operation> ::= (<MPLS_PUSH> <MPLS_LABEL> [<S_BIT>]
                              [<TOS_VALUE>] [<TTL_VALUE>]) |
                              (<MPLS_POP> [<TTL_ACTION>])

   <gre-header> ::= <GRE_IP_DESTINATION> <GRE_PROTOCOL_TYPE> [<GRE_KEY>]
   <vxlan-header> ::= (<ipv4-header> | <ipv6-header>)
                      [<VXLAN_IDENTIFIER>]
   <nvgre-header> ::= (<ipv4-header> | <ipv6-header>)
                      <VIRTUAL_SUBNET_ID>
                      [<FLOW_ID>]

   <nexthop-attributes> ::= [<NEXTHOP_ADDRESS_FAMILY>]
                            [<nexthop-flags>]
   <NEXTHOP_ADDRESS_FAMILY> ::= <IPV4> | <IPV6> | <ISO> | <IEEE MAC>
   <nexthop-flags> ::= [<NO_DECREMENT_TTL>] [<NO_PROPAGATE_TTL>]
   <nexthop-vendor-attributes> ::= <>
```

7. Using the RIB grammar

The RIB grammar is very generic and covers a variety of features. This section provides examples on using objects in the RIB grammar and examples to program certain use cases.

7.1. Using route preference and metric

Using route preference one can pre-install protection paths in the network. For example, if OSPF has a route preference of 10, then one can install a route with route preference of 20 to the same destination. The OSPF route will get precedence and will get installed in the FIB. When the OSPF route goes away (for any reason), the protection path will get installed in the FIB. If the hardware supports it, then the RIB manager can choose to pre-install both routes, with the OSPF nexthop getting preference.

Route preference can also be used to prevent denial of service attacks by installing routes with the best preference, which either drops the offending traffic or routes it to some monitoring/analysis station. Since the routes are installed with the best preference, they will supersede any route installed by any other protocol.

Route metric is used to disambiguate between 2 or more routes to the same destination with the same preference and in the same RIB. One usage of this is to install 2 routes, each with a different nexthop. The preferred nexthop is given a better metric than the other one. This results in traffic being forwarded to the preferred nexthop. If the preferred nexthop fails, then the RIB manager will automatically install a route to the other nexthop.

7.2. Using different nexthops types

The RIB grammar allows one to create a variety of nexthops. This section describes uses for certain types of nexthops.

7.2.1. Tunnel nexthops

A tunnel nexthop points to a tunnel of some kind. Traffic that goes over the tunnel gets encapsulated with the tunnel encap. Tunnel nexthops are useful for abstracting out details of the network, by having the traffic seamlessly route between network edges.

7.2.2. Replication lists

One can create a replication list for replication traffic to multiple destinations. The destinations, in turn, could be complex nexthops in themselves - at a level supported by the network device. Point to multipoint and broadcast are examples that involve replication.

A replication list (at the simplest level) can be represented as:

```
<nexthop-list> ::= <nexthop> [ <nexthop> ... ]
```

The above can be derived from the grammar as follows:

```
<nexthop-list> ::= <nexthop-list-member> [<nexthop-list-member> ...]
<nexthop-list> ::= <nexthop-chain> [<nexthop-chain> ...]
<nexthop-list> ::= <nexthop> [ <nexthop> ... ]
```

7.2.3. Weighted lists

A weighted list is used to load-balance traffic among a set of nexthops. From a modeling perspective, a weighted list is very similar to a replication list, with the difference that each member nexthop MUST have a LOAD_BALANCE_WEIGHT associated with it.

A weighted list (at the simplest level) can be represented as:

```
<nexthop-list> ::= (<nexthop> <LOAD_BALANCE_WEIGHT>)
                   [(<nexthop> <LOAD_BALANCE_WEIGHT>)... ]
```

The above can be derived from the grammar as follows:

```
<nexthop-list> ::= <nexthop-list-member> [<nexthop-list-member> ...]
<nexthop-list> ::= (<nexthop-chain> <nexthop-list-member-attributes>)
                   [(<nexthop-chain>
                     <nexthop-list-member-attributes>) ...]
<nexthop-list> ::= (<nexthop-chain> <LOAD_BALANCE_WEIGHT>)
                   [(<nexthop-chain> <LOAD_BALANCE_WEIGHT>) ... ]
<network-list> ::= (<nexthop> <LOAD_BALANCE_WEIGHT>)
                   [(<nexthop> <LOAD_BALANCE_WEIGHT>)... ]
```

7.2.4. Protection lists

Protection lists are similar to weighted lists. A protection list specifies a set of primary nexthops and a set of backup nexthops. The <PROTECTION_PREFERENCE> attribute indicates which nexthop is primary and which is backup.

A protection list can be represented as:

```
<nexthop-list> ::= (<nexthop> <PROTECTION_PREFERENCE>)
                   [(<nexthop> <PROTECTION_PREFERENCE>)... ]
```

A protection list can also be a weighted list. In other words, traffic can be load-balanced among the primary nexthops of a

protection list. In such a case, the list will look like:

```
<nexthop-list> ::= (<nexthop> <PROTECTION_PREFERENCE>
                              <LOAD_BALANCE_WEIGHT>)
                   [(<nexthop> <PROTECTION_PREFERENCE>
                               <LOAD_BALANCE_WEIGHT>)... ]
```

7.2.5. Nexthop chains

A nexthop chain is a nexthop that puts one or more headers on an outgoing packet. One example is a Pseudowire - which is MPLS over some transport (MPLS or GRE for instance). Another example is VxLAN over IP. A nexthop chain allows an external entity to break up the programming of the nexthop into independent pieces - one per encapsulation.

A simple example of MPLS over GRE can be represented as:

```
<nexthop-list> ::= (<MPLS> <mpls-header>) (<GRE> <gre-header>)
```

The above can be derived from the grammar as follows:

```
<nexthop-list> ::= <nexthop-list-member> [<nexthop-list-member> ...]
<nexthop-list> ::= <nexthop-chain>
<nexthop-list> ::= <nexthop> [ <nexthop> ... ]
<nexthop-list> ::= <tunnel-encap> (<nexthop> [ <nexthop> ...])
<nexthop-list> ::= <tunnel-encap> (<tunnel-encap>)
<nexthop-list> ::= (<MPLS> <mpls-header>) (<GRE> <gre-header>)
```

7.2.6. Lists of lists

Lists of lists is a complex construct. One example of usage of such a construct is to replicate traffic to multiple destinations, with high availability. In other words, for each destination you have a primary and backup nexthop (replication list) to ensure there is no traffic drop in case of a failure. So the outer list is a protection list and the inner lists are replication lists of primary/backup nexthops.

7.3. Performing multicast

IP multicast involves matching a packet on (S, G) or (*, G), where both S (source) and G (group) are IP prefixes. Following the match, the packet is replicated to one or more recipients. How the recipients subscribe to the multicast group is outside the scope of this document.

In PIM-based multicast, the packets are IP forwarded on an IP multicast tree. The downstream nodes on each point in the multicast tree is one or more IP addresses. These can be represented as a replication list (Section 7.2.2).

In MPLS-based multicast, the packets are forwarded on a point to multipoint (P2MP) label-switched path (LSP). The nexthop for a P2MP LSP can be represented in the nexthop grammar as a <logical-tunnel> (P2MP LSP identifier) or a replication list (Section 7.2.2) of <tunnel-encap>, with each tunnel encap representing a single mpls downstream nexthop.

7.4. Solving optimized exit control

In case of optimized exit control, a controller wants to control the edge device (and optionally control the outgoing interface on that edge device) that is used by a server to send traffic out. This can be easily achieved by having the controller program the edge router (Eg. 192.0.2.10) and the server along the following lines:

```
Server:
<route> ::= <rib-name> <match> (<edge-router>
                                          <edge-router-interface>)
<route> ::= <rib-name> <198.51.100.1/16>
                  (<MPLS> <mpls-header>)
                  (<GRE> <gre-header>)

<route> ::- <rib-name> <198.51.100.1/16>
                  (<MPLS_PUSH> <100>)
                  (<GRE> <192.0.2.10> <GRE_PROTOCOL_MPLS>)

Edge Router:
<route> ::= <mpls-rib> <mpls-route> <nexthop>
<route> ::= <mpls-rib> (<MPLS> <100>) <interface-10>
```

In the above case, the label 100 identifies the egress interface on the edge router.

8. RIB operations at scale

This section discusses the scale requirements for a RIB data-model. The RIB data-model should be able to handle large scale of operations, to enable deployment of RIB applications in large networks.

8.1. RIB reads

Bulking (grouping of multiple objects in a single message) MUST be supported when a network device sends RIB data to an external entity. Similarly the data model MUST enable a RIB client to request data in bulk from a network device.

8.2. RIB writes

Bulking (grouping of multiple write operations in a single message) MUST be supported when an external entity wants to write to the RIB. The response from the network device MUST include a return-code for each write operation in the bulk message.

8.3. RIB events and notifications

There can be cases where a single network event results in multiple events and/or notifications from the network device to an external entity. On the other hand, due to timing of multiple things happening at the same time, a network device might have to send multiple events and/or notifications to an external entity. The network device originated event/notification message MUST support bulking of multiple events and notifications in a single message.

9. Security Considerations

All interactions between a RIB manager and an external entity MUST be authenticated and authorized. The RIB manager MUST protect itself against a denial of service attack by a rogue external entity, by throttling request processing. A RIB manager MUST enforce limits on how much data can be programmed by an external entity and return error when such a limit is reached.

The RIB manager MUST expose a data-model that it implements. An external agent MUST send requests to the RIB manager that comply with the supported data-model. The data-model MUST specify the behavior of the RIB manager on handling of unsupported data requests.

10. IANA Considerations

This document does not generate any considerations for IANA.

11. Acknowledgements

The authors would like to thank the working group co-chairs and

reviewers on their comments and suggestions on this draft. The following people contributed to the design of the RIB model as part of the I2RS Interim meeting in April 2013 - Wes George, Chris Liljenstolpe, Jeff Tantsura, Sriganesh Kini, Susan Hares, Fabian Schneider and Nitin Bahadur.

12. References

12.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

12.2. Informative References

[I-D.atlas-i2rs-problem-statement]
Atlas, A., Nadeau, T., and D. Ward, "Interface to the Routing System Problem Statement", draft-atlas-i2rs-problem-statement-02 (work in progress), August 2013.

[I-D.hares-i2rs-use-case-vn-vc]
Hares, S., "Use Cases for Virtual Connections on Demand (VCoD) and Virtual Network on Demand using Interface to Routing System", draft-hares-i2rs-use-case-vn-vc-00 (work in progress), February 2013.

[I-D.white-i2rs-use-case]
White, R., Hares, S., and R. Fernando, "Use Cases for an Interface to the Routing System", draft-white-i2rs-use-case-00 (work in progress), February 2013.

[RFC3443] Agarwal, P. and B. Akyol, "Time To Live (TTL) Processing in Multi-Protocol Label Switching (MPLS) Networks", RFC 3443, January 2003.

[RFC4271] Rekhter, Y., Li, T., and S. Hares, "A Border Gateway Protocol 4 (BGP-4)", RFC 4271, January 2006.

[RFC4915] Psenak, P., Mirtorabi, S., Roy, A., Nguyen, L., and P. Pillay-Esnault, "Multi-Topology (MT) Routing in OSPF", RFC 4915, June 2007.

[RFC5065] Traina, P., McPherson, D., and J. Scudder, "Autonomous System Confederations for BGP", RFC 5065, August 2007.

[RFC5120] Przygienda, T., Shen, N., and N. Sheth, "M-ISIS: Multi Topology (MT) Routing in Intermediate System to Intermediate Systems (IS-ISs)", RFC 5120, February 2008.

[RFC5511] Farrel, A., "Routing Backus-Naur Form (RBNF): A Syntax Used to Form Encoding Rules in Various Routing Protocol Specifications", RFC 5511, April 2009.

Authors' Addresses

Nitin Bahadur (editor)
Juniper Networks, Inc.
1194 N. Mathilda Avenue
Sunnyvale, CA 94089
US

Phone: +1 408 745 2000
Email: nitinb@juniper.net
URI: www.juniper.net

Ron Folkes (editor)
Juniper Networks, Inc.
1194 N. Mathilda Avenue
Sunnyvale, CA 94089
US

Phone: +1 408 745 2000
Email: ronf@juniper.net
URI: www.juniper.net

Sriganesh Kini
Ericsson

Email: sriganesh.kini@ericsson.com

Jan Medved
Cisco

Email: jmedved@cisco.com