Network Working Group
Internet-Draft
Intended status: Informational
Expires: November 3, 2018

N. Bahadur, Ed.
Uber
S. Kini, Ed.

J. Medved
Cisco

May 2, 2018

Routing Information Base Info Model
draft-ietf-i2rs-rib-info-model-16

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 installs state into the hardware for packet forwarding. This draft specifies an information model for the RIB to enable defining a standardized data model, and it was used by the IETF's I2RS WG to design the I2RS RIB data model. It is being published to record the higher-level informational model decisions for RIBs so that other developers of RIBs may benefit from the design concepts.

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 November 3, 2018.

Copyright Notice

Table of Contents

1. Introduction 3
1.1. Conventions used in this document 5
2. RIB data 5
2.1. RIB definition 5
2.2. Routing instance 6
2.3. Route 7
2.4. Nexthop 9
2.4.1. Base nexthop 12
2.4.2. Derived nexthops 13
2.4.3. Nexthop indirection 15
3. Reading from the RIB 15
4. Writing to the RIB 15
5. Notifications 16
6. RIB grammar 16
6.1. Nexthop grammar explained 19
7. Using the RIB grammar 19
7.1. Using route preference 19
7.2. Using different nexthops types 20
7.2.1. Tunnel nexthops 20
7.2.2. Replication lists 20
7.2.3. Weighted lists 21
7.2.4. Protection 21
7.2.5. Nexthop chains 22
7.2.6. Lists of lists 23
7.3. Performing multicast 24
8. RIB operations at scale 25
8.1. RIB reads 25
8.2. RIB writes 25
8.3. RIB events and notifications 25
9. Security Considerations 25
10. IANA Considerations 26
11. Acknowledgements 26
12. References 26
12.1. Normative References 26
12.2. Informative References 27
Authors' Addresses 28

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 configuration information) populate the Routing information base (RIB) of the router. The RIB is managed by the RIB manager and the RIB manager provides a northbound 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.

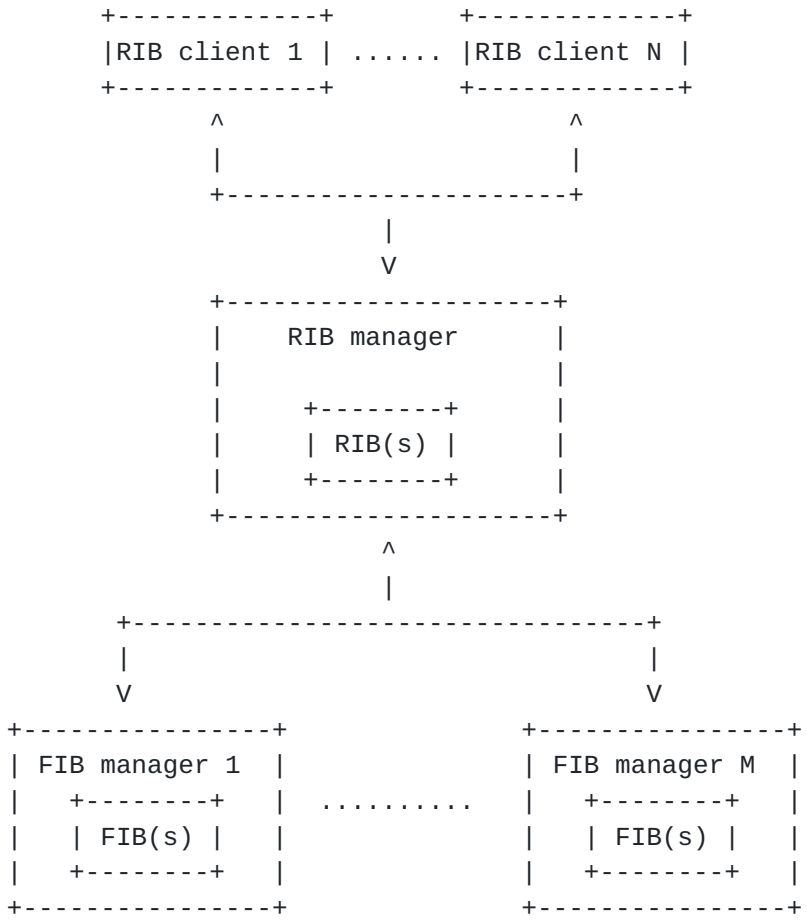

Figure 1: RIB manager, RIB clients, and FIB managers

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 [RFC7920].

Traditional network-device protocol-based RIB population suffices for most use cases where distributed network control is used. However there are use cases that the network operators currently address by configuring static routes, policies, and RIB import/export rules on the routers. There is also a growing list of use cases in which a network operator might want to program the RIB based on data unrelated to just routing (within that network's domain). Programming the RIB could be based on other information such as routing data in the adjacent domain or the load on storage and compute in the given domain. Or it could simply be a programmatic way of creating on-demand dynamic overlays (e.g., GRE tunnels) between compute hosts (without requiring the hosts to run traditional routing protocols). If there was a standardized publicly-documented, programmatic interface to a RIB, it would enable further networking applications that address a variety of use cases [RFC7920].

A programmatic interface to the RIB involves 2 types of operations - reading from the RIB and writing (adding/modifying/deleting) to the RIB.

In order to understand what is in a router's RIB, methods like per-protocol SNMP MIBs and screen scraping are 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 is vendor dependent. Building a RIB from per-protocol MIBs is error prone since the MIB data represent 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 a RIB client can be done today using static configuration mechanisms provided by router vendors. However the mix of what can be modified in the RIB varies from vendor to vendor and the method of configuring it is also vendor dependent. This makes it hard for a RIB client 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. That data model could then be used by a RIB client 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 a RIB client, to update the RIB client 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", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

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 in Figure 2. Please note that for ease of ASCII art representation this drawing shows a single routing-instance, a single RIB, and a single route. Sub-sections of this section describe the logical data nodes that should be contained within a RIB. Section 3 and Section 4 describe the high-level read and write operations.

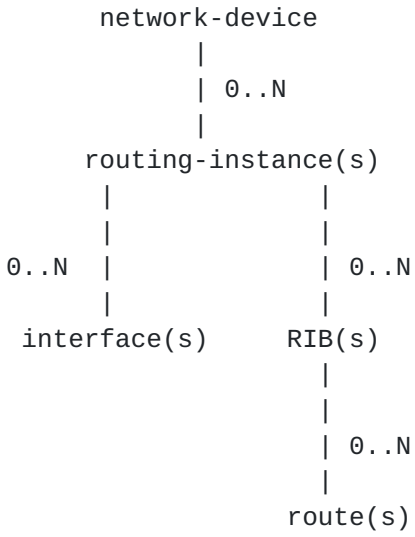

Figure 2: RIB model

2.1. RIB definition

A RIB, in the context of the RIB information model, is an entity that contains routes. It is identified by its name and is contained within a routing instance (Section 2.2). A network device MAY contain routing instances and each routing instance MAY contain RIBs.

The name MUST be unique within a routing instance. All routes in a given RIB MUST be of the same address family (e.g., IPv4). Each RIB MUST belong to a routing instance.

A routing instance may contain two or more RIBs of the same address family (e.g., IPv6). A typical case where this can be used is for multi-topology routing ([RFC4915], [RFC5120]).

Each RIB MAY be associated with an ENABLE_IP_RPF_CHECK attribute that enables REVERSE PATH FORWARDING (RPF) checks on all IP routes in that RIB. The 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. It allows different logical slices across a set of routers to communicate with each other. 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 MUST 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 MUST be unique across all routing instances in a given network device.

- 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. The list of RIBs can be an empty list.

A routing instance MAY contain the following 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 in 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: This 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.

A routing instance may be created purely for the purposes of packet processing and may not have any interfaces associated with it. For example, an incoming packet in routing instance A might have a nexthop of routing instance B and after packet processing in B, the nexthop might be routing instance C. Thus, routing instance B is not associated with any interface. And given that this routing instance does not do any control plane interaction with other network devices, a ROUTER_ID is also not needed.

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 3 represents the overall contents of a route. Please note that for ease of depiction in ASCII art only a single instance of the route attribute, match flags, or nexthop is depicted.

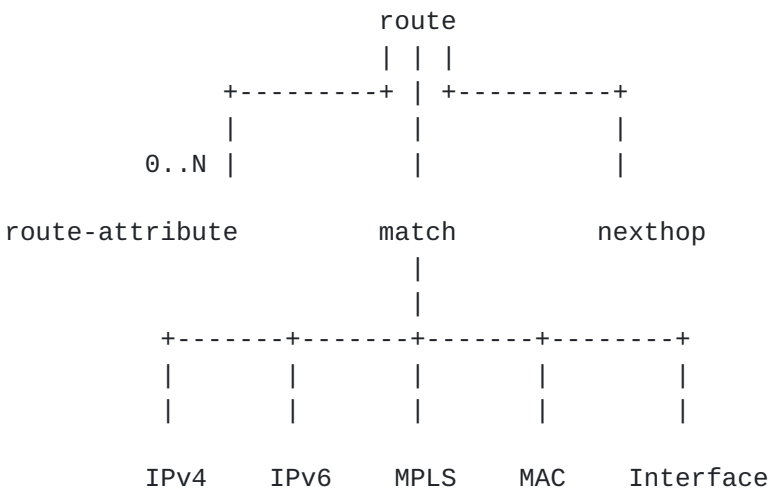

Figure 3: Route model

This document specifies the following match types:

- IPv4: Match on destination and/or source IP address in the IPv4 header
- IPv6: Match on destination and/or source IP address in the IPv6 header
- MPLS: Match on an MPLS label at the top of the MPLS label stack
- MAC: Match on MAC destination addresses in the Ethernet header
- Interface: Match on incoming interface of the packet

A route MAY be matched on one or more these match types by policy as either an "AND" (to restrict the number of routes) or an "OR" (to combine two filters).

Each route MUST have associated with it the following mandatory route attributes.

- ROUTE_PREFERENCE: This is a numerical value that allows for comparing routes from different protocols. 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 (or IPv6 2001:DB8::1/128) with a preference of 5.

If a controller programs a route for 192.0.2.1/32 (or IPv6 2001: DB8::1/128) with a preference of 2, then the controller's route will be preferred by the RIB manager. Preference should be used to dictate behavior. For more examples of preference, see Section 7.1.

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

- route-vendor-attributes: Vendors can specify vendor-specific attributes using this. The details of this attribute is outside the scope of this document.

Each route has associated with it a nexthop. Nexthop is described in Section 2.4.

Additional features to match multicast packets were considered (e.g., TTL of the packet to limit the range of a multicast group), but these were not added to this information model. Future RIB information models should investigate these multicast features.

2.4. Nexthop

A nexthop represents an object 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 has adequate information to send the outgoing packet to the destination by forwarding it on an interface to a directly connected neighbor. For example, a nexthop to a point-to-point interface or a nexthop to an IP address on an Ethernet interface has the nexthop resolved. An unresolved nexthop is something that requires the RIB manager to determine the final resolved nexthop. For example, a nexthop could be an IP address. The RIB manager would resolve how to reach that IP address, e.g., is the IP address reachable by regular IP forwarding or by an MPLS tunnel or by both. If the RIB manager cannot resolve the nexthop, then the nexthop remains in an unresolved state and is NOT a candidate for installation in the FIB. Future RIB events can cause an unresolved nexthop to get resolved (like that IP address being advertised by an IGP neighbor). Conversely, resolved nexthops can also become unresolved (e.g., in the case of a tunnel going down) and hence would no longer be candidates to be installed in the FIB.

When at least one of a route's nexthops is resolved, then the route can be used to forward packets. Such a route is considered eligible to be installed in the FIB and is henceforth referred to as a FIB-

eligible route. Conversely, when all the nexthops of a route are unresolved that route can no longer be used to forward packets. Such a route is considered ineligible to be installed in the FIB and is henceforth referred to as a FIB-ineligible route. The RIB information model allows a RIB client to program routes whose nexthops may be unresolved initially. Whenever an 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. For ease of ASCII art depiction, only a single instance of any component of the nexthop is shown in Figure 4.

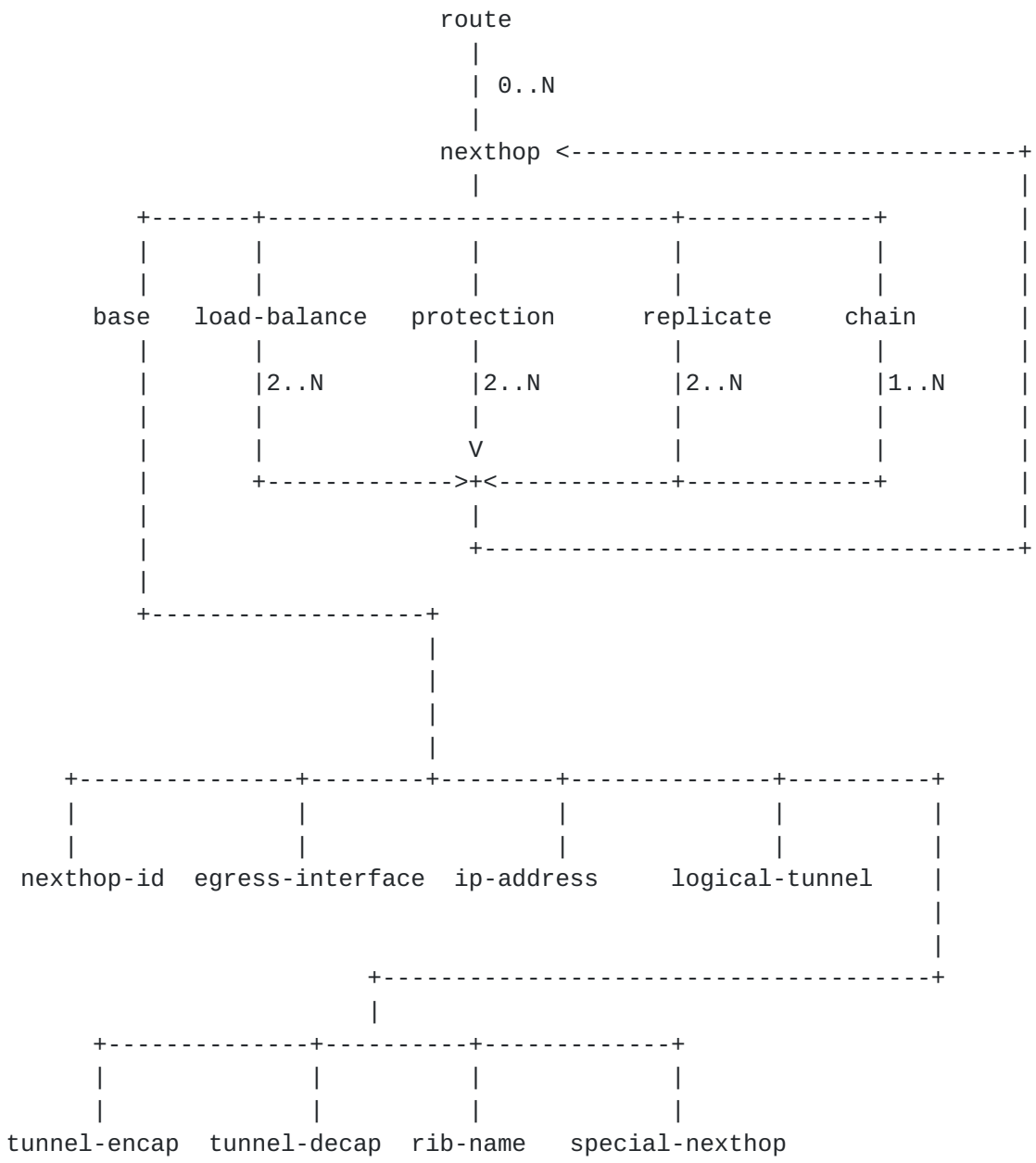

Figure 4: Nexthop model

This document specifies a very generic, extensible, and recursive grammar for nexthops. A nexthop can be a base nexthop or a derived nexthop. Section 2.4.1 details base nexthops and Section 2.4.2 explains various kinds of derived nexthops. There are certain special nexthops and those are described in Section 2.4.1.1. Lastly, Section 2.4.3 delves into nexthop indirection and it's use. Examples of when and how to use tunnel nexthops and derived nexthops are shown in Section 7.2.

2.4.1. Base nexthop

At the lowest level, a nexthop can be one of:

- Identifier: This is an identifier returned by the network device representing a nexthop. This can be used as a way of re-using a nexthop when programming derived nexthops.

- Interface nexthops - nexthops pointing to an interface. Various attributes associated with these nexthops are:

  * EGRESS_INTERFACE: This represents a physical, logical, or virtual interface on the network device. Address resolution must not be required on this interface. This interface may belong to any routing instance.

  * IP address: A route lookup on this IP address is done to determine the egress interface. Address resolution may be required depending on the interface.

    + An optional RIB name can also be specified to indicate the RIB in which the IP address is to be looked up. One can use the RIB name field to direct the packet from one domain into another domain. By default the RIB will be the same as the one that route belongs to.

  These attributes can be used in combination as follows:

  * EGRESS_INTERFACE and IP address: This can be used in cases, e.g., where the IP address is a link-local address.

  * EGRESS_INTERFACE and MAC address: The egress interface must be an Ethernet interface. Address resolution is not required for this nexthop.

- Tunnel nexthops - nexthops pointing to a tunnel. The types of tunnel nexthops are:

  * tunnel-encap: This can be an encapsulation representing an IP tunnel or MPLS tunnel or others as defined in this document. An optional egress interface can be chained to the tunnel-encap 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 address resolution for the IP packet.

  * tunnel-decap: This is to specify decapsulating a tunnel header. After decapsulation, further lookup on the packet can be done

via chaining it with another nexthop. The packet can also be sent out via an EGRESS_INTERFACE directly.

* logical-tunnel: This can be an 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. This indicates that the route lookup needs to continue in the specified RIB. This is a way to perform chained lookups.

Tunnel nexthops allow a RIB client to program static tunnel headers. There can be cases where the remote tunnel endpoint does not support dynamic signaling (e.g., no LDP support on a host) and in those cases the RIB client 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.

2.4.1.1. Special nexthops

Special nexthops are for performing specific well-defined functions (e.g., discard). 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.

2.4.2. Derived nexthops

Derived nexthops can be:

- Weighted lists - for load-balancing

- Preference lists - for protection using primary and backup

- Replication lists - list of nexthops to which to replicate a packet

- Nexthop chains - for chaining multiple operations or attaching multiple headers

- Lists of lists - recursive application of the above

Nexthop chains (See Section 7.2.5 for usage) is a way to perform multiple operations on a packet by logically combining them. For example, one can chain together "decapsulate MPLS header" and "send it out a specific EGRESS_INTERFACE". 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 an 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 or 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 headers chained together. The RIB data-model SHOULD provide a way to expose nexthop chaining capability supported by a given network device.

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 a RIB client to learn about the network device's capabilities.

2.4.2.1. 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 attributes are specified:

- NEXTHOP_PREFERENCE: This is used for protection schemes. It is an integer value between 1 and 99. A lower value indicates higher preference. To download a primary/standby pair to the FIB, the nexthops that are resolved and have the two highest preferences are selected. Each <NEXTHOP_PREFERENCE> should have a unique value within a <nexthop-protection> (Section 6).

- NEXTHOP_LB_WEIGHT: This is used for load-balancing. Each list member MUST be assigned a weight between 1 and 99. The weight determines the proportion of traffic to be sent over a nexthop used for forwarding as a ratio of the weight of this nexthop divided by the weights of all the nexthops of this route that are used for forwarding. 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. Note: A weight of 0 is special because of historical reasons.

2.4.3. Nexthop indirection

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 RIB client on request.

One example of usage of indirection is a nexthop that points to another network device (Eg. BGP peer). The returned nexthop identifier can then be used for programming routes to point to the this nexthop. Given that the RIB manager has created an indirection using the nexthop identifier, if the transport path to the network device (BGP peer) changes, that change in path will be seamless to the RIB client and all routes that point to that network device will automatically start going over the new transport path. Nexthop indirection using identifiers could be applied to not just unicast nexthops, but even to nexthops that contain chains and nested nexthops. See (Section 2.4.2) for examples.

3. Reading from the RIB

A RIB data-model MUST allow a RIB client to read entries for RIBs created by that entity. The network device administrator MAY allow reading of other RIBs by a RIB client 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. When sending data to a RIB client, 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 a RIB client to write entries for RIBs created by that entity. The network device administrator MAY allow writes to other RIBs by a RIB client 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 RIB client SHOULD try to write all dependencies of the object prior to sending that object. The

data-model SHOULD 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 can be modified. An object that can be modified 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 identified by a nexthop identifier should be unaffected when the contents of that nexthop changes.

5. Notifications

Asynchronous notifications are sent by the network device's RIB manager to a RIB client 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]. This grammar is intended to help the reader better understand Section 2 in order to derive a data model.

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


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

```
  <rib-list> ::= (<rib> ...)
  <rib> ::= <RIB_NAME> <address-family>
                       [<route> ... ]
                       [ENABLE_IP_RPF_CHECK]
  <address-family> ::= <IPV4_ADDRESS_FAMILY> | <IPV6_ADDRESS_FAMILY> |
                        <MPLS_ADDRESS_FAMILY> | <IEEE_MAC_ADDRESS_FAMILY>


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


  <match> ::= <IPV4> <ipv4-route> | <IPV6> <ipv6-route> |
              <MPLS> <MPLS_LABEL> | <IEEE_MAC> <MAC_ADDRESS> |
              <INTERFACE> <INTERFACE_IDENTIFIER>
  <route-type> ::= <IPV4> | <IPV6> | <MPLS> | <IEEE_MAC> | <INTERFACE>


  <ipv4-route> ::= <ip-route-type>
                   (<destination-ipv4-address> | <source-ipv4-address> |
                    (<destination-ipv4-address> <source-ipv4-address>))
  <destination-ipv4-address> ::= <ipv4-prefix>
  <source-ipv4-address> ::= <ipv4-prefix>
  <ipv4-prefix> ::= <IPV4_ADDRESS> <IPV4_PREFIX_LENGTH>


  <ipv6-route> ::= <ip-route-type>
                   (<destination-ipv6-address> | <source-ipv6-address> |
                    (<destination-ipv6-address> <source-ipv6-address>))
  <destination-ipv6-address> ::= <ipv6-prefix>
  <source-ipv6-address> ::= <ipv6-prefix>
  <ipv6-prefix> ::= <IPV6_ADDRESS> <IPV6_PREFIX_LENGTH>
  <ip-route-type> ::= <SRC> | <DEST> | <DEST_SRC>


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

  <address-family-route-attributes> ::= <ip-route-attributes> |
                                        <mpls-route-attributes> |
                                        <ethernet-route-attributes>
  <ip-route-attributes> ::= <>
  <mpls-route-attributes> ::= <>
  <ethernet-route-attributes> ::= <>
  <route-vendor-attributes> ::= <>
```

```
  <nexthop> ::= <nexthop-base> |
                (<NEXTHOP_LOAD_BALANCE> <nexthop-lb>) |
                (<NEXTHOP_PROTECTION> <nexthop-protection>) |
                (<NEXTHOP_REPLICATE> <nexthop-replicate>) |
                <nexthop-chain>

  <nexthop-base> ::= <NEXTHOP_ID> |
                     <nexthop-special> |
                     <EGRESS_INTERFACE> |
                     <ipv4-address> | <ipv6-address> |
                     (<EGRESS_INTERFACE>
                         (<ipv4-address> | <ipv6-address>)) |
                     (<EGRESS_INTERFACE> <IEEE_MAC_ADDRESS>) |
                     <tunnel-encap> | <tunnel-decap> |
                     <logical-tunnel> |
                     <RIB_NAME>)

  <EGRESS_INTERFACE> ::= <INTERFACE_IDENTIFIER>

  <nexthop-special> ::= <DISCARD> | <DISCARD_WITH_ERROR> |
                        (<RECEIVE> [<COS_VALUE>])


  <nexthop-lb> ::= <NEXTHOP_LB_WEIGHT> <nexthop>
                   (<NEXTHOP_LB_WEIGHT> <nexthop) ...


  <nexthop-protection> = <NEXTHOP_PREFERENCE> <nexthop>
                         (<NEXTHOP_PREFERENCE> <nexthop>)...

  <nexthop-replicate> ::= <nexthop> <nexthop> ...


  <nexthop-chain> ::= <nexthop> ...


  <logical-tunnel> ::= <tunnel-type> <TUNNEL_NAME>
  <tunnel-type> ::= <IPV4> | <IPV6> | <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_SWAP> <IN_LABEL> <OUT_LABEL>
                                        [<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>]

<tunnel-decap> ::= ((<IPV4> <IPV4_DECAP> [<TTL_ACTION>]) |
                    (<IPV6> <IPV6_DECAP> [<HOP_LIMIT_ACTION>]) |
                    (<MPLS> <MPLS_POP> [<TTL_ACTION>]))
```

Figure 5: RIB rBNF grammar

6.1. Nexthop grammar explained

A nexthop is used to specify the next network element to forward the traffic to. It is also used to specify how the traffic should be load-balanced, protected using preference, or multicast using replication. This is explicitly specified in the grammar. The nexthop has recursion built-in to address complex use cases like the one defined in Section 7.2.6.

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

Using route preference a client can pre-install alternate paths in the network. For example, if OSPF has a route preference of 10, then another client 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 is withdrawn, the

alternate path will get installed in the FIB.

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.

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. At the end of a tunnel, the tunnel will get decapsulated. Thus the grammar supports two kinds of operations, one for encapsulation and another for decapsulation.

7.2.2. Replication lists

One can create a replication list for replicating traffic to multiple destinations. The destinations, in turn, could be derived 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> ::= <NEXTHOP_REPLICATE> <nexthop> [ <nexthop> ... ]
```

The above can be derived from the grammar as follows:

```
<nexthop> ::= <nexthop-replicate>
<nexthop> ::= <NEXTHOP_REPLICATE> <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 NEXTHOP_LB_WEIGHT associated with it.

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

```
<nexthop> ::= <NEXTHOP_LOAD_BALANCE> (<nexthop> <NEXTHOP_LB_WEIGHT>)
                  [(<nexthop> <NEXTHOP_LB_WEIGHT>)... ]
```

The above can be derived from the grammar as follows:

```
<nexthop> ::= <nexthop-lb>
<nexthop> ::= <NEXTHOP_LOAD_BALANCE>
                <NEXTHOP_LB_WEIGHT> <nexthop>
                (<NEXTHOP_LB_WEIGHT> <nexthop>) ...
<nexthop> ::= <NEXTHOP_LOAD_BALANCE> (<NEXTHOP_LB_WEIGHT> <nexthop>)
                (<NEXTHOP_LB_WEIGHT> <nexthop>) ...
```

7.2.4. Protection

A primary/backup protection can be represented as:

```
<nexthop> ::= <NEXTHOP_PROTECTION> <1> <interface-primary>
                                   <2> <interface-backup>)
```

The above can be derived from the grammar as follows:

```
<nexthop> ::= <nexthop-protection>
<nexthop> ::= <NEXTHOP_PROTECTION> (<NEXTHOP_PREFERENCE> <nexthop>
                    (<NEXTHOP_PREFERENCE> <nexthop>)...)
<nexthop> ::= <NEXTHOP_PROTECTION> (<NEXTHOP_PREFERENCE> <nexthop>
                    (<NEXTHOP_PREFERENCE> <nexthop>))
<nexthop> ::= <NEXTHOP_PROTECTION> ((<NEXTHOP_PREFERENCE> <nexthop-base>
                    (<NEXTHOP_PREFERENCE> <nexthop-base>))
<nexthop> ::= <NEXTHOP_PROTECTION> (<1> <interface-primary>
                    (<2> <interface-backup>))
```

Traffic can be load-balanced among multiple primary nexthops and a single backup. In such a case, the nexthop will look like:

```
<nexthop> ::= <NEXTHOP_PROTECTION> (<1>
              (<NEXTHOP_LOAD_BALANCE>
               (<NEXTHOP_LB_WEIGHT> <nexthop-base>
               (<NEXTHOP_LB_WEIGHT> <nexthop-base>) ...))
                <2> <nexthop-base>)
```

A backup can also have another backup. In such a case, the list will look like:

```
<nexthop> ::= <NEXTHOP_PROTECTION> (<1> <nexthop>
              <2> <NEXTHOP_PROTECTION>(<1> <nexthop> <2> <nexthop>))
```

7.2.5. Nexthop chains

A nexthop chain is a way to perform multiple operations on a packet by logically combining them. For example, when a VPN packet comes on the WAN interface and has to be forwarded to the correct VPN interface, one needs to POP the VPN label before sending the packet out. Using a nexthop chain, one can chain together "pop MPLS header" and "send it out a specific EGRESS_INTERFACE".

The above example can be derived from the grammar as follows:

```
<nexthop-chain> ::= <nexthop> <nexthop>
<nexthop-chain> ::= <nexthop-base> <nexthop-base>
<nexthop-chain> ::= <tunnel-decap> <EGRESS_INTERFACE>
<nexthop-chain> ::= (<MPLS> <MPLS_POP>) <interface-outgoing>
```

Elements in a nexthop-chain are evaluated left to right.

A nexthop chain can also be used to put 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 thus allows a RIB client 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-chain> ::= (<MPLS> <mpls-header>) (<GRE> <gre-header>)
                    <interface-outgoing>
```

The above can be derived from the grammar as follows:

```
<nexthop-chain> ::= <nexthop> <nexthop> <nexthop>
<nexthop-chain> ::= <nexthop-base> <nexthop-base> <nexthop-base>
<nexthop-chain> ::= <tunnel-encap> <tunnel-encap> <EGRESS_INTERFACE>
<nexthop-chain> ::= (<MPLS> <mpls-header>) (<GRE> <gre-header>)
                    <interface-outgoing>
```

7.2.6. Lists of lists

Lists of lists is a derived construct. One example of usage of such a construct is to replicate traffic to multiple destinations, with load balancing. In other words for each branch of the replication tree, there are multiple interfaces on which traffic needs to be load-balanced on. So the outer list is a replication list for multicast and the inner lists are weighted lists for load balancing. Let's take an example of a network element has to replicate traffic to two other network elements. Traffic to the first network element should be load balanced equally over two interfaces outgoing-1-1 and outgoing-1-2. Traffic to the second network element should be load balanced over three interfaces outgoing-2-1, outgoing-2-2 and outgoing-2-3 in the ratio 20:20:60.

This can be derived from the grammar as follows:

```
<nexthop> ::= <nexthop-replicate>
<nexthop> ::= <NEXTHOP_REPLICATE> (<nexthop> <nexthop>...)
<nexthop> ::= <NEXTHOP_REPLICATE> (<nexthop> <nexthop>)
<nexthop> ::= <NEXTHOP_REPLICATE> ((<NEXTHOP_LOAD_BALANCE> <nexthop-lb>)
              (<NEXTHOP_LOAD_BALANCE> <nexthop-lb>))
<nexthop> ::= <NEXTHOP_REPLICATE> ((<NEXTHOP_LOAD_BALANCE>
              (<NEXTHOP_LB_WEIGHT> <nexthop>
              (<NEXTHOP_LB_WEIGHT> <nexthop>) ...))
               ((<NEXTHOP_LOAD_BALANCE>
                (<NEXTHOP_LB_WEIGHT> <nexthop>
                (<NEXTHOP_LB_WEIGHT> <nexthop>) ...))
<nexthop> ::= <NEXTHOP_REPLICATE> ((<NEXTHOP_LOAD_BALANCE>
              (<NEXTHOP_LB_WEIGHT> <nexthop>
               (<NEXTHOP_LB_WEIGHT> <nexthop>)))
                ((<NEXTHOP_LOAD_BALANCE>
                (<NEXTHOP_LB_WEIGHT> <nexthop>
                (<NEXTHOP_LB_WEIGHT> <nexthop>)
                (<NEXTHOP_LB_WEIGHT> <nexthop>)))
<nexthop> ::= <NEXTHOP_REPLICATE> ((<NEXTHOP_LOAD_BALANCE>
               (<NEXTHOP_LB_WEIGHT> <nexthop>)
               (<NEXTHOP_LB_WEIGHT> <nexthop>)))
               ((<NEXTHOP_LOAD_BALANCE>
               (<NEXTHOP_LB_WEIGHT> <nexthop>)
               (<NEXTHOP_LB_WEIGHT> <nexthop>)
               (<NEXTHOP_LB_WEIGHT> <nexthop>)))
<nexthop> ::= <NEXTHOP_REPLICATE>
               ((<NEXTHOP_LOAD_BALANCE>
                 (50 <outgoing-1-1>)
                 (50 <outgoing-1-2>)))
                ((<NEXTHOP_LOAD_BALANCE>
                  (20 <outgoing-2-1>)
                  (20 <outgoing-2-2>)
                  (60 <outgoing-2-3>)))
```

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.

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 a RIB client. 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 a RIB client 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 a RIB client. 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 a RIB client. The network device originated event/ notification message MUST support bulking of multiple events and notifications in a single message.

9. Security Considerations

The Informational module specified in this document defines a schema for data models that are designed to be accessed via network management protocols such as NETCONF [RFC6241] or RESTCONF [RFC8040]. The lowest NETCONF layer is the secure transport layer, and the mandatory-to-implement secure transport is Secure Shell (SSH)

[RFC6242]. The lowest RESTCONF layer is HTTPS, and the mandatory-to-implement secure transport is TLS [RFC5246].

The NETCONF access control model [RFC8341] provides the means to restrict access for particular NETCONF or RESTCONF users to a preconfigured subset of all available NETCONF or RESTCONF protocol operations and content.

The RIB info model specifies read and write operations to network devices. These network devices might be considered sensitive or vulnerable in some network environments. Write operations to these network devices without proper protection can have a negative effect on network operations. Due to this factor, it is recommended that data models also consider the following in their design:

- Require utilization of the authentication and authorization features of the NETCONF or RESTCONF suite of protocols.
- Augment the limits on how much data can be written or updated by a remote entity built to include enough protection for a RIB model.
- Expose the specific RIB model implemented via NETCONF/RESTCONF data models.

10. IANA Considerations

This document does not generate any considerations for IANA.

11. Acknowledgements

The authors would like to thank Ron Folkes, Jeffrey Zhang, the working group co-chairs, and reviewers for 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, Susan Hares, and Fabian Schneider.

12. References

12.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

12.2. Informative References

[RFC4915] Psenak, P., Mirtorabi, S., Roy, A., Nguyen, L., and P. Pillay-Esnault, "Multi-Topology (MT) Routing in OSPF", RFC 4915, DOI 10.17487/RFC4915, June 2007, <https://www.rfc-editor.org/info/rfc4915>.

[RFC5120] Przygienda, T., Shen, N., and N. Sheth, "M-ISIS: Multi Topology (MT) Routing in Intermediate System to Intermediate Systems (IS-ISs)", RFC 5120, DOI 10.17487/RFC5120, February 2008, <https://www.rfc-editor.org/info/rfc5120>.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <https://www.rfc-editor.org/info/rfc5246>.

[RFC5511] Farrel, A., "Routing Backus-Naur Form (RBNF): A Syntax Used to Form Encoding Rules in Various Routing Protocol Specifications", RFC 5511, DOI 10.17487/RFC5511, April 2009, <https://www.rfc-editor.org/info/rfc5511>.

[RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011, <https://www.rfc-editor.org/info/rfc6241>.

[RFC6242] Wasserman, M., "Using the NETCONF Protocol over Secure Shell (SSH)", RFC 6242, DOI 10.17487/RFC6242, June 2011, <https://www.rfc-editor.org/info/rfc6242>.

[RFC7920] Atlas, A., Ed., Nadeau, T., Ed., and D. Ward, "Problem Statement for the Interface to the Routing System", RFC 7920, DOI 10.17487/RFC7920, June 2016, <https://www.rfc-editor.org/info/rfc7920>.

[RFC8040] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", RFC 8040, DOI 10.17487/RFC8040, January 2017, <https://www.rfc-editor.org/info/rfc8040>.

[RFC8341] Bierman, A. and M. Bjorklund, "Network Configuration Access Control Model", STD 91, RFC 8341, DOI 10.17487/RFC8341, March 2018,

<https://www.rfc-editor.org/info/rfc8341>.

Authors' Addresses

Nitin Bahadur (editor)
Uber
900 Arastradero Rd
Palo Alto, CA 94304
US

Email: nitin_bahadur@yahoo.com

Sriganesh Kini (editor)

Email: sriganeshkini@gmail.com

Jan Medved
Cisco

Email: jmedved@cisco.com