

Internet Engineering Task Force  
Internet-Draft  
Intended status: Informational  
Expires: April 23, 2014

J. Medved  
S. Previdi  
Cisco Systems, Inc.  
V. Lopez  
Telefonica I+D  
S. Amante

October 20, 2013

Topology API Use Cases  
draft-amante-i2rs-topology-use-cases-01

Abstract

This document describes use cases for gathering routing, forwarding and policy information, (hereafter referred to as topology information), about the network. It describes several applications that need to view the topology of the underlying physical or logical network. This document further demonstrates a need for a "Topology Manager" and related functions that collects topology data from network elements and other data sources, coalesces the collected data into a coherent view of the overall network topology, and normalizes the network topology view for use by clients -- namely, applications that consume topology information.

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 April 23, 2014.

## Copyright Notice

Copyright (c) 2013 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

1. Introduction . . . . .	2
1.1. Statistics Collection . . . . .	5
1.2. Inventory Collection . . . . .	5
1.3. Requirements Language . . . . .	6
2. Terminology . . . . .	6
3. The Orchestration, Collection & Presentation Framework . . . . .	7
3.1. Overview . . . . .	7
3.2. The Topology Manager . . . . .	8
3.3. The Policy Manager . . . . .	10
3.4. Orchestration Manager . . . . .	10
4. Use Cases . . . . .	11
4.1. Virtualized Views of the Network . . . . .	11
4.1.1. Capacity Planning and Traffic Engineering . . . . .	11
4.1.1.1. Present Mode of Operation . . . . .	12
4.1.1.2. Proposed Mode of Operation . . . . .	12
4.1.2. Services Provisioning . . . . .	14
4.1.3. Troubleshooting & Monitoring . . . . .	14
4.2. Virtual Network Topology Manager (VNTM) . . . . .	15
4.3. Path Computation Element (PCE) . . . . .	16
4.4. ALTO Server . . . . .	17
5. Acknowledgements . . . . .	18
6. IANA Considerations . . . . .	18
7. Security Considerations . . . . .	18
8. References . . . . .	18
8.1. Normative References . . . . .	18
8.2. Informative References . . . . .	19
Authors' Addresses . . . . .	20

## 1. Introduction

In today's networks, a variety of applications, such as Traffic Engineering, Capacity Planning, Security Auditing or Services Provisioning (for example, Virtual Private Networks), have a common need to acquire and consume network topology information. Unfortunately, all of these applications are (typically) vertically integrated: each uses its own proprietary normalized view of the network and proprietary data collectors, interpreters and adapters, which speak a variety of protocols, (SNMP, CLI, SQL, etc.) directly to network elements and to back-office systems. While some of the topological information can be distributed using routing protocols, unfortunately it is not desirable for some of these applications to understand or participate in routing protocols.

This approach is incredibly inefficient for several reasons. First, developers must write duplicate 'network discovery' functions, which then become challenging to maintain over time, particularly as/when new equipment are first introduced to the network. Second, since there is no common "vocabulary" to describe various components in the network, such as physical links, logical links, or IP prefixes, each application has its own data model. To solve this, some solutions have distributed this information in the normalized form of routing distribution. However, this information still does not contain "inactive" topological information, thus not containing information considered to be part of a network's inventory.

These limitations lead to applications being unable to easily exchange information with each other. For example, applications cannot share changes with each other that are (to be) applied to the physical and/or logical network, such as installation of new physical links, or deployment of security ACL's. Each application must frequently poll network elements and other data sources to ensure that it has a consistent representation of the network so that it can carry out its particular domain-specific tasks. In other cases, applications that cannot speak routing protocols must use proprietary CLI or other management interfaces which represent the topological information in non-standard formats or worse, semantic models.

Overall, the software architecture described above at best results in inefficient use of both software developer resources and network resources, and at worst, it results in some applications simply not having access to this information.

Figure 1 is an illustration of how individual applications collect data from the underlying network. Applications retrieve inventory, network topology, state and statistics information by communicating directly with underlying Network Elements as well as with intermediary proxies of the information. In addition, applications transmit changes required of a Network Element's configuration and/or

state directly to individual Network Elements, (most commonly using CLI or Netconf). It is important to note that the "data models" or semantics of this information contained within Network Elements are largely proprietary with respect to most configuration and state information, hence why a proprietary CLI is often the only choice to reflect changes in a NE's configuration or state. This remains the case even when standards-based mechanisms such as Netconf are used which provide a standard syntax model, but still often lack due to the proprietary semantics associated with the internal representation of the information.

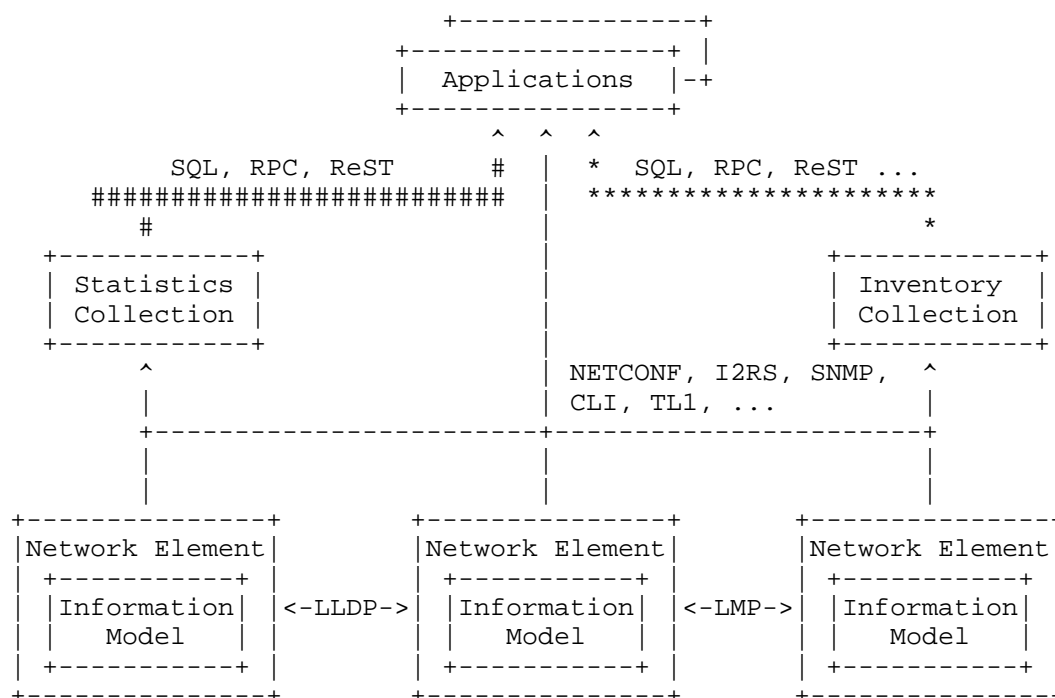


Figure 1: Applications getting topology data

Figure 1 shows how current management interfaces such as NETCONF, SNMP, CLI, etc. are used to transmit or receive information to/from various Network Elements. The figure also shows that protocols such as LLDP and LMP participate in topology discovery, specifically to discover adjacent network elements.

The following sections describe the "Statistics Collection" and "Inventory Collection" functions.

### 1.1. Statistics Collection

In Figure 1, "Statistics Collection" is a dedicated infrastructure that collects statistics from Network Elements. It periodically (for example, every 5-minutes) polls Network Elements for octets transferred per interface, per LSP, etc. Collected statistics are stored and collated within a statistics data warehouse. Applications typically query the statistics data warehouse rather than poll Network Elements directly to get the appropriate set of link utilization figures for their analysis.

### 1.2. Inventory Collection

"Inventory Collection" is a network function responsible for collecting component and state information directly from Network Elements, as well as for storing inventory information about physical network assets that are not retrievable from Network Elements. The collected data is hereafter referred to as the 'Inventory Asset Database. Examples of information collected from Network Elements are: interface up/down status, the type of SFP/XFP optics inserted into physical port, etc.

The Inventory Collection function may use SNMP and CLI to acquire inventory information from Network Elements. The information housed in the Inventory Manager is retrieved by applications via a variety of protocols: SQL, RPC, REST etc. Inventory information, retrieved from Network Elements, is periodically updated in the Inventory Collection system to reflect changes in the physical and/or logical network assets. The polling interval to retrieve updated information is varied depending on scaling constraints of the Inventory Collection systems and expected intervals at which changes to the physical and/or logical assets are expected to occur.

Examples of changes in network inventory that need be learned by the Inventory Collection function are as follows:

- o Discovery of new Network Elements. These elements may or may not be actively used in the network (i.e.: provisioned but not yet activated).
- o Insertion or removal of line cards or other modules, such as optics modules, during service or equipment provisioning.
- o Changes made to a specific Network Element through a management interface by a field technician.
- o Indication of an NE's physical location and associated cable run list, at the time of installation.

- o Insertion of removal of cables that result in dynamic discovery of a new or lost adjacent neighbor, etc.

### 1.3. Requirements Language

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 RFC 2119 [RFC2119]

## 2. Terminology

The following briefly defines some of the terminology used within this document.

**Inventory Manager** is a function that collects Network Element inventory and state information directly from Network Elements and from associated offline inventory databases. Inventory information may only be visible at a specific network layer; for example, a physical link is visible within the IGP, but a Layer-2 switch through which the physical link traverses is unknown to the Layer-3 IGP.

**Policy Manager** is a function that attaches metadata to network components/attributes. Such metadata may include security, routing, L2 VLAN ID, IP numbering, etc. policies, which enable the Topology Manager to:

- \* Assemble a normalized view of the network for clients (or upper-layer applications
- \* Allow clients (or upper-layer applications) access to information collected from various network layers and/or network components, etc.

The Policy Manager function may be a sub-component of the Topology Manager or it may be a standalone function.

**Topology Manager** is a function that collects topological information from a variety of sources in the network and provides a normalized view of the network topology to clients and/or higher-layer applications.

**Orchestration Manager** is a function that stitches together resources, such as compute or storage, and/or services with the network or vice-versa. To realize a complete service, the Orchestration Manager relies on capabilities provided by the other "Managers" listed above.

Normalized Topology Information Model is an open, standards-based information model of the network topology.

Information Model Abstraction: The notion that one is able to represent the same set of elements in an information model at different levels of "focus" in order to limit the amount of information exchanged in order to convey this information.

Multi-Layer Topology: Topology is commonly referred to using the OSI protocol layering model. For example, Layer 3 represents routed topologies that typically use IPv4 or IPv6 addresses. It is envisioned that, eventually, multiple layers of the network may be represented in a single, normalized view of the network to certain applications, (i.e.: Capacity Planning, Traffic Engineering, etc.)

Network Element (NE) refers to a network device that typically is addressable (but not always), or a host. It is sometimes referred to as a 'Node'.

Links: Every NE contains at least 1 link. These are used to connect the NE to other NEs in the network. Links may be in a variety of states, such as up, down, administratively down, internally testing, or dormant. Links are often synonymous with network ports on NEs.

### 3. The Orchestration, Collection & Presentation Framework

#### 3.1. Overview

Section 1 demonstrates the need for a network function that would provide a common, standards-based topology view to applications. Such topology collection/management/presentation function would be a part of a wider framework that should also include policy management and orchestration. The framework is shown in Figure 2.

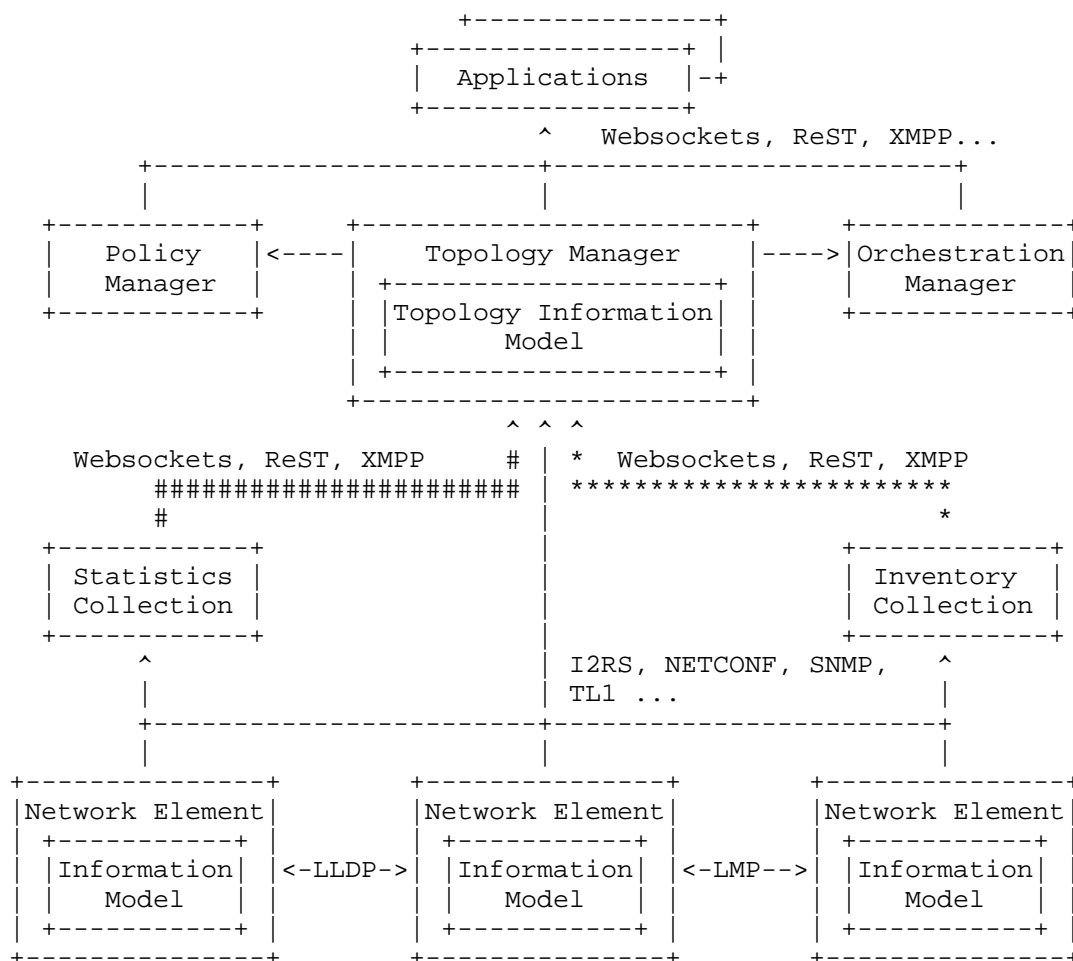


Figure 2: Topology Manager

The following sections describe in detail the Topology Manager, Policy Manager and Orchestration Manager functions.

### 3.2. The Topology Manager

The Topology Manager is a function that collects topological information from a variety of sources in the network and provides a cohesive, abstracted view of the network topology to clients and/or higher-layer applications. The topology view is based on a standards-based, normalized topology information model.

Topology information sources can be:



- o The "live" Layer 3 IGP or an equivalent mechanism that provides information about links that are components of the active topology. Active topology links are present in the Link State Database (LSDB) and are eligible for forwarding. Layer 3 IGP information can be obtained by listening to IGP updates flooded through an IGP domain, or from Network Elements.
- o The Inventory Collection system that provides information for network components not visible within the Layer 3 IGP's LSDB, (i.e.: links or nodes, or properties of those links or nodes, at lower layers of the network).
- o The Statistics Collection system that provides traffic information, such as traffic demands or link utilizations.

The Topology Manager provides topology information to Clients or higher-layer applications via a northbound interface, such as ReST, Websockets, or XMPP.

The Topology Manager will contain topology information for multiple layers of the network: Transport, Ethernet and IP/MPLS, as well as information for multiple Layer 3 IGP areas and multiple Autonomous Systems (ASes). The topology information can be used by higher-level applications, such as Traffic Engineering, Capacity Planning and Provisioning. Such applications are typically used to design, augment and optimize IP/MPLS networks, and require knowledge of underlying Shared Risk Link Groups (SRLG) within the Transport and/or Ethernet layers of the network.

The Topology Manager must be able to discover Network Elements that are not visible in the "live" L3 IGP's Link State Database (LSDB). Such Network Elements can either be inactive, or active but invisible in the L3 LSDB (e.g.: L2 Ethernet switches, ROADMs, or Network Elements that are in an underlying transport network).

In addition static inventory information collected from the Inventory Manager, the Topology Manager will also collect dynamic inventory information. For example, Network Elements utilize various Link Layer Discovery Protocols (i.e.: LLDP, LMP, etc.) to automatically identify adjacent nodes and ports. This information can be pushed to or pulled by the Topology Manager in order to create an accurate representation of the physical topology of the network

### 3.3. The Policy Manager

The Policy Manager is the function used to enforce and program policies applicable to network component/attribute data. Policy enforcement is a network-wide function that can be consumed by various Network Elements and services, including the Inventory Manager, the Topology Manager and other Network Elements. Such policies are likely to encompass the following:

- o Logical Identifier Numbering Policies
  - \* Correlation of IP prefix to link based on link type, such as P-P, P-PE, or PE-CE.
  - \* Correlation of IP Prefix to IGP Area
  - \* Layer-2 VLAN ID assignments, etc.
- o Routing Configuration Policies
  - \* OSPF Area or IS-IS Net-ID to Node (Type) Correlation
  - \* BGP routing policies, such as nodes designated for injection of aggregate routes, max-prefix policies, or AFI/SAFI to node correlation..
- o Security Policies
  - \* Access Control Lists
  - \* Rate-limiting
- o Network Component/Attribute Data Access Policies. Client's (upper-layer application) access to Network Components/Attributes contained in the "Inventory Manager" as well as Policies contained within the "Policy Manager" itself.

The Policy Manager function may be either a sub-component of the Topology or Orchestration Manager or a standalone component.

### 3.4. Orchestration Manager

The Orchestration Manager provides the ability to stitch together resources (such as compute or storage) and/or services with the network or vice-versa. Examples of 'generic' services may include the following:

- o Application-specific Load Balancing

- o Application-specific Network (Bandwidth) Optimization
- o Application or End-User specific Class-of-Service
- o Application or End-User specific Network Access Control

The above services could then enable coupling of resources with the network to realize the following:

- o Network Optimization: Creation and Migration of Virtual Machines (VM's) so they are adjacent to storage in the same DataCenter.
- o Network Access Control: Coupling of available (generic) compute nodes within the appropriate point of the data-path to perform firewall, NAT, etc. functions on data traffic.

The Orchestration Manager will exchange information models with the Topology Manager, the Policy Manager and the Inventory Manager. In addition, the Orchestration Manager must support publish and subscribe capabilities to those functions, as well as to Clients.

The Orchestration Manager may receive requests from Clients (applications) for immediate access to specific network resources. However, Clients may request to schedule future appointments to reserve appropriate network resources when, for example, a special event is scheduled to start and end.

Finally, the Orchestration Manager should have the flexibility to determine what network layer(s) may be able to satisfy a given Client's request, based on constraints received from the Client as well as constraints learned from the Policy and Topology Managers. This could allow the Orchestration Manager to, for example, satisfy a given service request for a given Client using the optical network (via OTN service) if there is insufficient IP/MPLS capacity at the specific moment the Client's request is received.

The operational model is shown in the following figure.

TBD.

Figure 3: Overall Reference Model

#### 4. Use Cases

##### 4.1. Virtualized Views of the Network

###### 4.1.1. Capacity Planning and Traffic Engineering

#### 4.1.1.1. Present Mode of Operation

When performing Traffic Engineering and/or Capacity Planning of an IP /MPLS network, it is important to account for SRLG's that exist within the underlying physical, optical and Ethernet networks. Currently, it's quite common to take "snapshots" at infrequent intervals that comprise the inventory data of the underlying physical and optical layer networks. This inventory data is then normalized to conform to data import requirements of sometimes separate Traffic Engineering and/or Capacity Planning tools. This process is error-prone and inefficient, particularly as the underlying network inventory information changes due to introduction of new network element makes or models, line cards, capabilities, etc..

The present mode of operation is inefficient with respect to Software Development, Capacity Planning and Traffic Engineering resources. Due to this inefficiency, the underlying physical network inventory information (containing SRLG and corresponding critical network assets information) is not updated frequently, thus exposing the network to, at minimum, inefficient utilization and, at worst, critical impairments.

#### 4.1.1.2. Proposed Mode of Operation

First, the Inventory Manager will extract inventory information from network elements and associated inventory databases. Information extracted from inventory databases will include physical cross-connects and other information that is not available directly from network elements. Standards-based information models and associated vocabulary will be required to represent not only components inside or directly connected to network elements, but also to represent components of a physical layer path (i.e.: cross-connect panels, etc.) The inventory data will comprise the complete set of inactive and active network components.

Second, the Topology Manager will augment the inventory information with topology information obtained from Network Elements and other sources, and provide an IGP-based view of the active topology of the network. The Topology Manager will also include non-topology dynamic information from IGPs, such as Available Bandwidth, Reserved Bandwidth, Traffic Engineering (TE) attributes associated with links, etc.

Finally, the Statistics Collector will collect utilization statistics from Network Elements, and archive and aggregate them in a statistics data warehouse. Selected statistics and other dynamic data may be distributed through IGP routing protocols ([I-D.ietf-isis-te-metric-extensions] and

[I-D.ietf-ospf-te-metric-extensions]) and then collected at the Statistics Collection Function via BGP-LS ([I-D.ietf-idr-ls-distribution]). Statistics summaries then will be exposed in normalized information models to the Topology Manager, which can use them to, for example, build trended utilization models to forecast expected changes to physical and logical network components.

It is important to recognize that extracting topology information from the network solely from Network Elements and IGP's (IS-IS TE or OSPF TE), is inadequate for this use case. First, IGP's only expose the active components (e.g. vertices of the SPF tree) of the IP network, and are not aware of "hidden" or inactive interfaces within IP/MPLS network elements, such as unused line cards or ports. IGP's are also not aware of components that reside at a layer lower than IP /MPLS, such as Ethernet switches, or Optical transport systems. Second, IGP's only convey SRLG information that have been first applied within a router's configurations, either manually or programmatically. As mentioned previously, this SRLG information in the IP/MPLS network is subject to being infrequently updated and, as a result, may inadequately account for critical, underlying network fate sharing properties that are necessary to properly design resilient circuits and/or paths through the network.

Once the Topology Manager has assembled a normalized view of the topology and metadata associated with each component of the topology, it can expose this information via its northbound API to the Capacity Planning and Traffic Engineering applications. The applications only require generalized information about nodes and links that comprise the network, e.g.: links used to interconnect nodes, SRLG information (from the underlying network), utilization rates of each link over some period of time, etc.

Note that any client/application that understands the Topology Manager's northbound API and its topology information model can communicate with the Topology Manager. Note also that topology information may be provided by Network Elements from different vendors, which may use different information models. If a Client wanted to retrieve topology information directly from Network Elements, it would have to translate and normalize these different representations.

A Traffic Engineering application may run a variety of CSPF algorithms that create a list of TE tunnels that globally optimize the packing efficiency of physical links throughout the network. The TE tunnels are then programmed into the network either directly or through a controller. Programming of TE tunnels into the network is outside the scope of this document.

A Capacity Planning application may run a variety of algorithms the result of which is a list of new inventory that is required for purchase or redeployment, as well as associated work orders for field technicians to augment the network for expected growth.

#### 4.1.2. Services Provisioning

Beyond Capacity Planning and Traffic Engineering applications, having a normalized view of just the IP/MPLS layer of the network is still very important for other mission critical applications, such as Security Auditing and IP/MPLS Services Provisioning, (e.g.: L2VPN, L3VPN, etc.). With respect to the latter, these types of applications should not need a detailed understanding of, for example, SRLG information, assuming that the underlying MPLS Tunnel LSP's are known to account for the resiliency requirements of all services that ride over them. Nonetheless, for both types of applications it is critical to have a common and up-to-date normalized view of the IP/MPLS network to, for example, instantiate new services at optimal locations in the network, or to validate proper ACL configuration to protect associated routing, signaling and management protocols on the network.

A VPN Service Provisioning application must perform the following resource selection operations:

- o Identify Service PE's in all markets/cities where the customer has identified they want service
- o Identify one or more existing Services PE's in each city with connectivity to the access network(s), e.g.: SONET/TDM, used to deliver the PE-CE tail circuits to the Service's PE.
- o Determine that the Services PE have available capacity on both the PE-CE access interface and its uplinks to terminate the tail circuit

The VPN Provisioning application would iteratively query the Topology Manager to narrow down the scope of resources to the set of Services PEs with the appropriate uplink bandwidth and access circuit capability plus capacity to realize the requested VPN service. Once the VPN Provisioning application has a candidate list of resources it requests programming of the Service PE's and associated access circuits to set up a customer's VPN service into the network. Programming of Service PEs is outside the scope of this document.

#### 4.1.3. Troubleshooting & Monitoring

Once the Topology Manager has a normalized view of several layers of the network, it can expose a rich set of data to network operators who are performing diagnosis, troubleshooting and repairs on the network. Specifically, there is a need to (rapidly) assemble a current, accurate and comprehensive network diagram of a L2VPN or L3VPN service for a particular customer when either: a) attempting to diagnose a service fault/error; or, b) attempting to augment the customer's existing service. Information that may be assembled into a comprehensive picture could include physical and logical components related specifically to that customer's service, i.e.: VLAN's or channels used by the PE-CE access circuits, CoS policies, historical PE-CE circuit utilization, etc. The Topology Manager would assemble this information, on behalf of each of the network elements and other data sources in and associated with the network, and would present this information in a vendor-independent data model to applications to be displayed allowing the operator (or, potentially, the customer through a SP's Web portal) to visualize the information.

#### 4.2. Virtual Network Topology Manager (VNTM)

Virtual Network Topology Manager (VNTM) is in charge of managing the Virtual Network Topology (VNT), as defined in [RFC5623]. VNT is defined in [RFC5212] as a set of one or more LSPs in one or more lower-layer networks that provides information for efficient path handling in an upper-layer network.

The maintenance of virtual topology is a complicated task. VNTM have to decide which are the nodes to be interconnected in the lower-layer to fulfill the resource requirements of the upper-layer. This means to create a topology to cope with all demands of the upper layer without wasting resources in the underlying network. Once the decision is made, some actions have to be taken in the network elements of the layers so the new LSPs are provisioned. Moreover, VNT has to release unwanted resources, so they can be available in the lower-layer network for other uses.

VNTM does not have to solve all previous problems in all scenarios. As defined in [RFC5623] in the PCE-VNTM cooperation model, PCE is computing the paths in the higher layer and when there is not enough resources in the VNT, PCE requests to the VNTM for a new path in the VNT. VNTM checks PCE request using internal policies to check whether this request can be take into account or not. VNTM requests the egress node in the upper layer to set-up the path in the lower layer. However, the VNTM can actively modify the VNT based on the policies and network status without waiting to an explicit PCE request.

Regarding the provisioning phase, VNTM may have to directly talk with an NMS to set-up the connection [RFC5623] or it can delegate this function to the provisioning manager [I-D.farrkingel-pce-abno-architecture].

The aim of this document is not to categorize all implementation options for VNTM, but to present the necessity to retrieve topological information to perform its functions. The VNTM may require the topologies of the lower and/or upper layer and even the inter-layer relation between the upper and lower layer nodes, to decide which is the optimal VNT.

#### 4.3. Path Computation Element (PCE)

As described in [RFC4655] a PCE can be used to compute MPLS-TE paths within a "domain" (such as an IGP area) or across multiple domains (such as a multi-area AS, or multiple ASes).

- o Within a single area, the PCE offers enhanced computational power that may not be available on individual routers, sophisticated policy control and algorithms, and coordination of computation across the whole area.
- o If a router wants to compute a MPLS-TE path across IGP areas its own TED lacks visibility of the complete topology. That means that the router cannot determine the end-to-end path, and cannot even select the right exit router (Area Border Router - ABR) for an optimal path. This is an issue for large-scale networks that need to segment their core networks into distinct areas, but which still want to take advantage of MPLS-TE.

The PCE presents a computation server that may have visibility into more than one IGP area or AS, or may cooperate with other PCEs to perform distributed path computation. The PCE needs access to the topology and the Traffic Engineering Database (TED) for the area(s) it serves, but [RFC4655] does not describe how this is achieved. Many implementations make the PCE a passive participant in the IGP so that it can learn the latest state of the network, but this may be sub-optimal when the network is subject to a high degree of churn, or when the PCE is responsible for multiple areas.

The following figure shows how a PCE can get its TED information using a Topology Server.



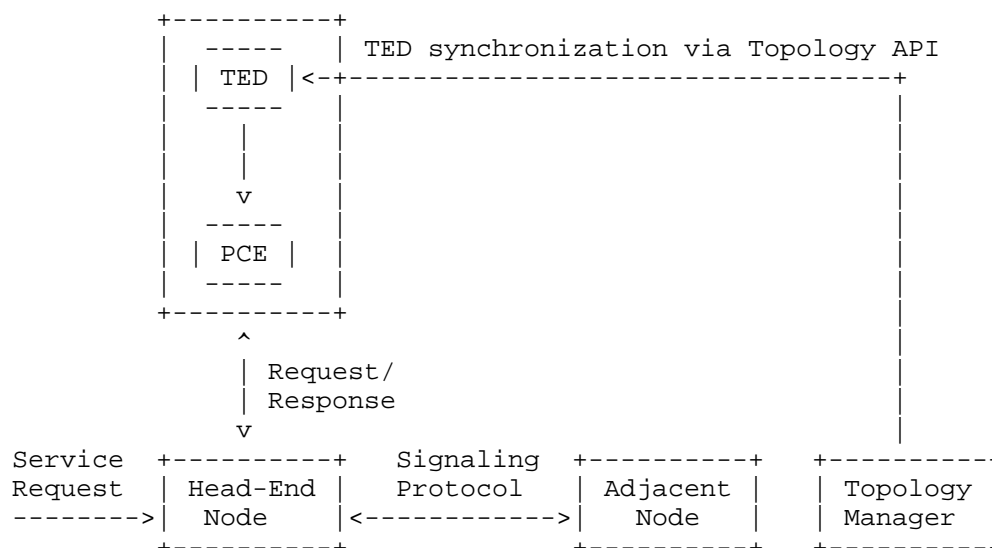


Figure 4: Topology use case: Path Computation Element

#### 4.4. ALTO Server

An ALTO Server [RFC5693] is an entity that generates an abstracted network topology and provides it to network-aware applications over a web service based API. Example applications are p2p clients or trackers, or CDNs. The abstracted network topology comes in the form of two maps: a Network Map that specifies allocation of prefixes to PIDs, and a Cost Map that specifies the cost between PIDs listed in the Network Map. For more details, see [I-D.ietf-alto-protocol].

ALTO abstract network topologies can be auto-generated from the physical topology of the underlying network. The generation would typically be based on policies and rules set by the operator. Both prefix and TE data are required: prefix data is required to generate ALTO Network Maps, TE (topology) data is required to generate ALTO Cost Maps. Prefix data is carried and originated in BGP, TE data is originated and carried in an IGP. The mechanism defined in this document provides a single interface through which an ALTO Server can retrieve all the necessary prefix and network topology data from the underlying network. Note an ALTO Server can use other mechanisms to get network data, for example, peering with multiple IGP and BGP Speakers.

The following figure shows how an ALTO Server can get network topology information from the underlying network using the Topology API.

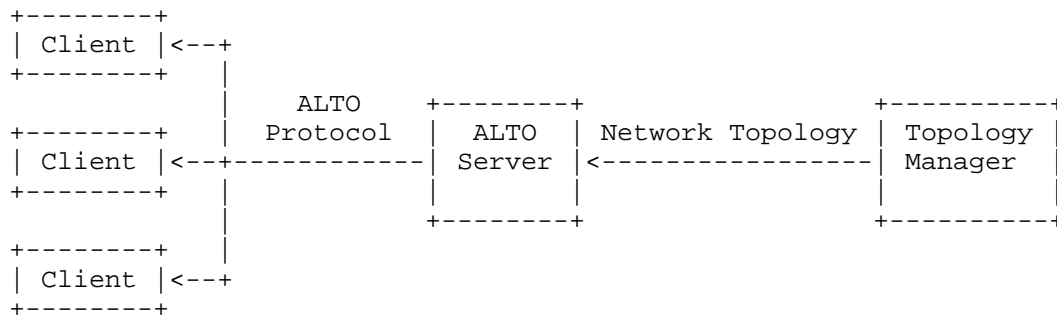


Figure 5: Topology use case: ALTO Server

## 5. Acknowledgements

The authors wish to thank Alia Atlas, Dave Ward, Hannes Gredler, Stefano Previdi for their valuable contributions and feedback to this draft.

## 6. IANA Considerations

This memo includes no request to IANA.

## 7. Security Considerations

At the moment, the Use Cases covered in this document apply specifically to a single Service Provider or Enterprise network. Therefore, network administrations should take appropriate precautions to ensure appropriate access controls exist so that only internal applications and end-users have physical or logical access to the Topology Manager. This should be similar to precautions that are already taken by Network Administrators to secure their existing Network Management, OSS and BSS systems.

As this work evolves, it will be important to determine the appropriate granularity of access controls in terms of what individuals or groups may have read and/or write access to various types of information contained with the Topology Manager. It would be ideal, if these access control mechanisms were centralized within the Topology Manager itself.

## 8. References

### 8.1. Normative References

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

## 8.2. Informative References

- [I-D.farrkingel-pce-abno-architecture]  
King, D. and A. Farrel, "A PCE-based Architecture for Application-based Network Operations", draft-farrkingel-pce-abno-architecture-05 (work in progress), July 2013.
- [I-D.ietf-alto-protocol]  
Alimi, R., Penno, R., and Y. Yang, "ALTO Protocol", draft-ietf-alto-protocol-17 (work in progress), July 2013.
- [I-D.ietf-idr-ls-distribution]  
Gredler, H., Medved, J., Previdi, S., Farrel, A., and S. Ray, "North-Bound Distribution of Link-State and TE Information using BGP", draft-ietf-idr-ls-distribution-03 (work in progress), May 2013.
- [I-D.ietf-isis-te-metric-extensions]  
Previdi, S., Giacalone, S., Ward, D., Drake, J., Atlas, A., and C. Filsfils, "IS-IS Traffic Engineering (TE) Metric Extensions", draft-ietf-isis-te-metric-extensions-00 (work in progress), June 2013.
- [I-D.ietf-ospf-te-metric-extensions]  
Giacalone, S., Ward, D., Drake, J., Atlas, A., and S. Previdi, "OSPF Traffic Engineering (TE) Metric Extensions", draft-ietf-ospf-te-metric-extensions-04 (work in progress), June 2013.
- [I-D.ietf-pce-stateful-pce]  
Crabbe, E., Medved, J., Minei, I., and R. Varga, "PCEP Extensions for Stateful PCE", draft-ietf-pce-stateful-pce-05 (work in progress), July 2013.
- [RFC4655] Farrel, A., Vasseur, J., and J. Ash, "A Path Computation Element (PCE)-Based Architecture", RFC 4655, August 2006.
- [RFC5212] Shiimoto, K., Papadimitriou, D., Le Roux, J.L., Vigoureux, M., and D. Brungard, "Requirements for GMPLS-Based Multi-Region and Multi-Layer Networks (MRN/MLN)", RFC 5212, July 2008.
- [RFC5623] Oki, E., Takeda, T., Le Roux, J.L., and A. Farrel, "Framework for PCE-Based Inter-Layer MPLS and GMPLS Traffic Engineering", RFC 5623, September 2009.

[RFC5693] Seedorf, J. and E. Burger, "Application-Layer Traffic Optimization (ALTO) Problem Statement", RFC 5693, October 2009.

Authors' Addresses

Jan Medved  
Cisco Systems, Inc.  
170 West Tasman Drive  
San Jose, CA 95134  
USA

Email: [jmedved@cisco.com](mailto:jmedved@cisco.com)

Stefano Previdi  
Cisco Systems, Inc.  
170, West Tasman Drive  
San Jose, CA 95134  
USA

Email: [sprevidi@cisco.com](mailto:sprevidi@cisco.com)

Victor Lopez  
Telefonica I+D  
c/ Don Ramon de la Cruz 84  
Madrid 28006  
Spain

Email: [vlopez@tid.es](mailto:vlopez@tid.es)

Shane Amante

I2RS  
Internet-Draft  
Intended status: Informational  
Expires: April 01, 2014

J. Clarke  
G. Salgueiro  
C. Pignataro  
Cisco  
September 28, 2013

Interface to the Routing System (I2RS) Traceability: Framework and  
Information Model  
draft-clarke-i2rs-traceability-00

Abstract

This document describes a framework for traceability in the Interface to the Routing System (I2RS) and information model for that framework. It specifies the motivation, requirements, use cases, and defines an information model for recording interactions between elements implementing the I2RS protocol. This framework provides a consistent tracing interface for components implementing the I2RS architecture to record what was done, by which component, and when. It aims to improve the management of I2RS implementations, and can be used for troubleshooting, auditing, forensics, and accounting purposes.

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 April 01, 2014.

Copyright Notice

Copyright (c) 2013 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents

(<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

1. Introduction . . . . .	2
2. Terminology and Conventions . . . . .	3
3. Motivation and Use Cases . . . . .	3
4. Information Model . . . . .	4
4.1. I2RS Traceability Framework . . . . .	4
4.2. I2RS Trace Log Mandatory Fields . . . . .	5
4.3. I2RS Trace Log Extensibility and Optional Fields . . . . .	6
4.4. I2RS Trace Log Syntax . . . . .	6
5. Examples . . . . .	7
6. Operational Guidance . . . . .	8
6.1. Trace Log Creation . . . . .	8
6.2. Trace Log Temporary Storage . . . . .	8
6.3. Trace Log Rotation . . . . .	9
6.4. Trace Log Retrieval . . . . .	9
6.4.1. Retrieval Via Syslog . . . . .	9
6.4.2. Retrieval Via I2RS Information Collection . . . . .	9
6.4.3. Retrieval Via I2RS Pub-Sub . . . . .	10
7. IANA Considerations . . . . .	10
8. Security Considerations . . . . .	10
9. References . . . . .	10
9.1. Normative References . . . . .	10
9.2. Informative References . . . . .	11
Authors' Addresses . . . . .	11

## 1. Introduction

The architecture for the Interface to the Routing System ([I-D.ietf-i2rs-architecture]) specifies that I2RS Clients wishing to retrieve or change routing state on a routing element MUST authenticate to an I2RS Agent. The I2RS Client will have a unique identity it provides for authentication, and should provide another, opaque identifier for applications (or actors) communicating through it. The programming of routing state will produce in a return code containing the results of the specified operation and associated reason(s) for the result. All of this is critical information to be used for understanding the history of I2RS interactions.

This document specifies requirements, use cases, and describes the information model for traceability attributes that must be collected and logged by I2RS Agents in order to effectively record I2RS interactions. In this context, effective troubleshooting means being able to identify what operation was performed by a specific I2RS Client, what was the result of the operation, and when that operation was performed.

Discussions about the retention of the data logged as part of I2RS traceability, while important, are outside of the scope of this document.

## 2. Terminology and Conventions

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].

The architecture specification for I2RS [I-D.ietf-i2rs-architecture] defines additional terms used in this document that are specific to the I2RS domain, such as "I2RS Agent", "I2RS Client", etc. The reader is expected to be familiar with the terminology and concepts defined in [I-D.ietf-i2rs-architecture].

The IP addresses used in the example in this document correspond to the documentation address blocks 192.0.2.0/24 (TEST-NET-1), 198.51.100.0/24 (TEST-NET-2) and 203.0.113.0/24 (TEST-NET-3) as described in [RFC5737].

## 3. Motivation and Use Cases

As networks grow, so too does the scale and complexity of routing systems. I2RS offers a standard, programmatic interface allowing improved automation and more granular control over an increasingly dynamic routing and signaling state. This ability to automate even complex policy-based controls highlights the need for an equally scalable traceability function to provide event-level granularity of the routing system compliant with the requirements of I2RS (Section 5 of [I-D.ietf-i2rs-problem-statement]).

An obvious motivation for I2RS traceability is the need to troubleshoot and identify root-causes of problems in these increasingly complex routing systems. For example, since I2RS is a high-throughput multi-channel, full duplex and highly responsive interface, I2RS Clients may be performing a large number of operations on I2RS Agents concurrently or at nearly the same time and quite possibly in very rapid succession. As these many changes are made, the network reacts accordingly. These changes might lead to a

race condition, performance issues, data loss, or disruption of services. In order to isolate the root cause of these issues it is critical that a network operator or administrator has visibility into what changes were made via I2RS at a specific time.

Some network environments have strong auditing requirements for configuration and runtime changes. Other environments have internal policies to save logging information for operational considerations. These requirements therefore demand that I2RS provides an account of changes made to network element routing systems.

As I2RS becomes increasingly pervasive in routing environments, a traceability model offers significant advantages and facilitates the following use cases:

- o Automated event correlation, trend analysis, and anomaly detection.
- o Trace log storage for offline (manual or tools) analysis.
- o Improved accounting of routing system transactions.
- o Standardized structured data format for writing common tools.
- o Common reference for automated testing and incident reporting.
- o Real-time monitoring and troubleshooting.
- o Enhanced network audit, management and forensic analysis capabilities.

#### 4. Information Model

##### 4.1. I2RS Traceability Framework

This section describes a framework for I2RS traceability based on the I2RS Architecture. Some notable elements on the architecture are highlighted herein.

The interaction between the optional northbound actor, I2RS Client, I2RS Agent, the Routing System and the data captured in the I2RS trace log is shown in Figure 1.



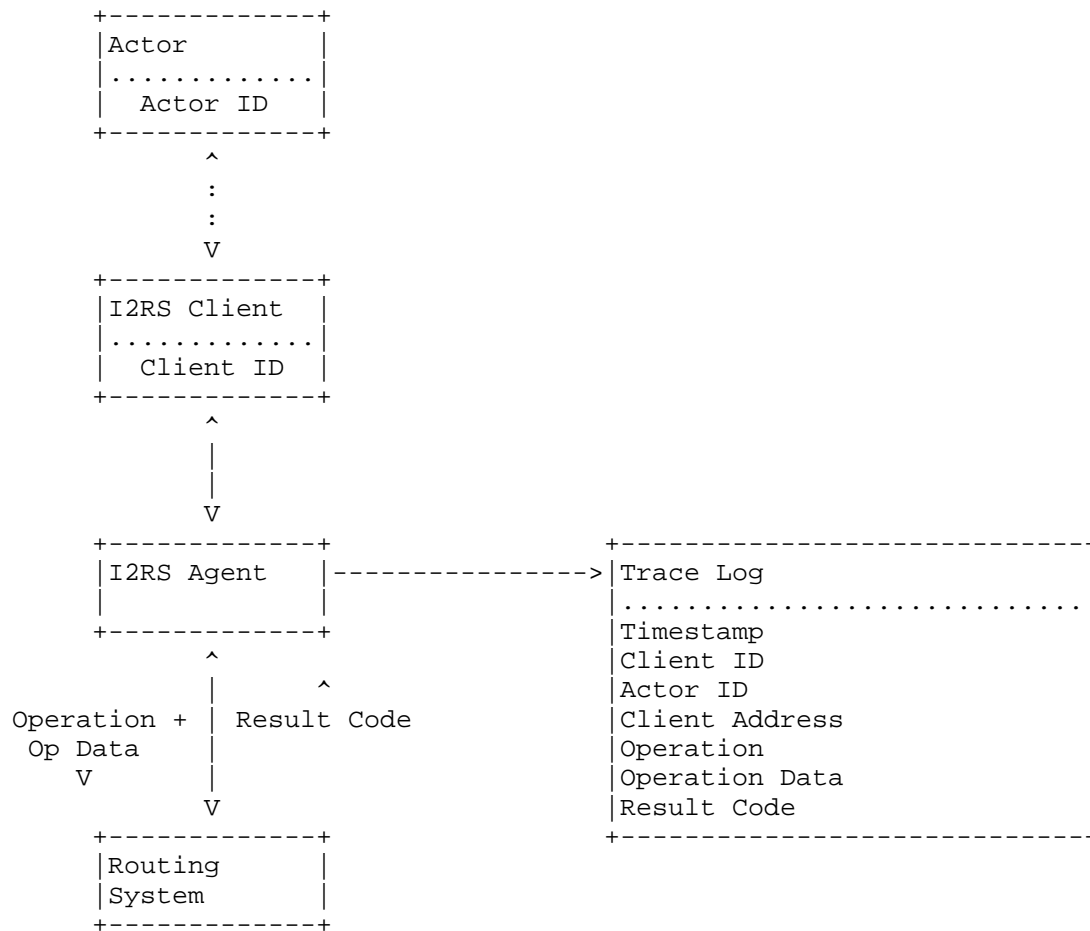


Figure 1: I2RS Interaction Trace Log Capture

#### 4.2. I2RS Trace Log Mandatory Fields

In order to ensure that each I2RS interaction can be properly traced back to the Client that made the request at a specific point in time, the following information **MUST** be collected and stored by the Agent.

The list below describes the fields captured in the I2RS trace log.

**Timestamp:** The specific time, adhering to [RFC3339] format, at which the I2RS transaction occurred. Given that many I2RS transactions can occur in rapid succession, the use of fractional seconds **MUST** be used to provide adequate granularity.

**Client Identifier:** The I2RS Client identifier used to authenticate the Client to the I2RS Agent.

**Actor Identifier:** This is an opaque identifier that may be known to the Client from a northbound controlling application. This is used to trace the northbound actor driving the actions of the Client. The Client may not provide this identifier to the Agent if there is no external actor driving the Client. However, this field **MUST** be logged. If the Client does not provide an actor ID, then the Agent **MUST** log an empty field.

**Client Address:** This is the network address of the client that connected to the Agent. For example, this may be an IPv4 or IPv6 address. [Note: will I2RS support interactions that have no network address? If so this field will need to be updated.]

**Operation:** This is the I2RS operation performed. For example, this may be an add route operation if a route is being inserted into a routing table.

**Operation Data:** This field comprises the data passed to the Agent to complete the desired operation. For example, if the operation is a route add operation, the Operation Data would include the route prefix, prefix length, and next hop information to be inserted as well as the specific routing table to which the route will be added. The operation data can also include interface information. Some operations **MAY** not provide operation data, and in those cases this field **MUST** be logged as a NULL string.

**Result Code:** This field holds the result of the operation. In the case of RIB operations, this **MUST** be the return code as specified in Section 4 of [I-D.nitinb-i2rs-rib-info-model]. The operation may not complete with a result code in the case of a timeout. If the operation fails to complete, it **MUST** still log the attempted operation with an appropriate result code (e.g., a result code indicating a timeout).

#### 4.3. I2RS Trace Log Extensibility and Optional Fields

[NOTE: This section is TBD based on further development of I2RS WG milestones.]

#### 4.4. I2RS Trace Log Syntax

The following describes the trace log information model using Augmented Backus-Naur Form (ABNF) syntax [RFC5234]:

```
i2rs-trace-log = timestamp client-id actor-id client-addr operation
                  operation-data result-code
client-id       = uuid
actor-id       = byte-string
byte-string    = *( %x01-09 / %x0B-0C / %x0E-FF )
                  ; any byte except NUL, CR or LF
client-addr    = IP-literal / IPv4address
operation      = ALPHA *( ALPHA / "_" / "-" / "(" / ")" )
operation-data = *VCHAR
result-code    = 1*( ALPHA / DIGIT / "-" / "_" / "(" / ")" )
```

The ABNF syntax rules <IP-literal>, <IPv4address>, and <sub-delims> are specified in [RFC3986]. The core rules <DIGIT>, <ALPHA> and <VCHAR> are used as described in Appendix B of [RFC5234].

Here, <uuid> is specified in [RFC4122]. The idea of using a UUID for the Client identifier ensures the ID is unique not just in the scope of the current I2RS Agent, but across Agents as well. This ensures that two clients that are unaware of each other will not allocate the same Client ID. That does not preclude two Clients acting as one for purposes of high availability from sharing the same UUID as generated by one one of the Clients. [Note: the format of the Client ID has not yet been decided by the I2RS WG. This is subject to change.]

The <timestamp> field is defined in [RFC3339]. As stated in Section 4.2 the fractional second format MUST be used to provide proper granularity.

The values for <operation>, <operation-data> and <result-code> are suggestions as the protocol has not been defined yet. By making these human-readable (as opposed to opcodes) the log becomes more easily consumable by operators and administrators trying to troubleshoot issues relating to I2RS. Even in cases where the operations or codes might appear as opcodes on the wire, their textual translations MUST be included in the log. The opcodes themselves MAY appear in parentheses after the textual representation.

## 5. Examples

Here is a proposed sample of what the fields might look like in an I2RS trace log. This is only an early proposal. These values are subject to change.

```
Timestamp:      2013-09-03T12:00:01.21+00:00
Client ID:      5CEF1870-0326-11E2-A21F-0800200C9A66
Actor ID:       com.example.RoutingApp
Client Address: 192.0.2.2
Operation:      ROUTE_ADD
Operation Data: PREFIX 203.0.113.0 PREFIX-LEN 24 NEXT-HOP
                198.51.100.1
Result Code:    SUCCESS(0)
```

## 6. Operational Guidance

Specific operational procedures regarding temporary log storage, rollover, retrieval, and access of I2RS trace logs is out of scope for this document. Organizations employing I2RS trace logging are responsible for establishing proper operational procedures that are appropriately suited to their specific requirements and operating environment. In this section we only provide fundamental and generalized operational guidelines that are implementation-independent.

### 6.1. Trace Log Creation

The I2RS Agent interacts with the Routing and Signaling functions of the Routing Element. Since the I2RS Agent is responsible for actually making the routing changes on the associated network device, it creates and maintains a log of transactions that can be retrieved to troubleshoot I2RS-related impact to the network.

### 6.2. Trace Log Temporary Storage

The trace information may be temporarily stored either in an in-memory buffer or as a file local to the Agent. Care should be given to the number of I2RS transactions expected on a given agent so that the appropriate storage medium is used and to maximize the effectiveness of the log while not impacting the performance and health of the Agent. Section 6.3 talks about rotating the trace log in order to preserve the transaction history without exhausting Agent or network device resources. It is perfectly acceptable, therefore, to use both an in-memory buffer for recent transactions while rotating or archiving older transactions to a local file.

It is outside the scope of this document to specify the implementation details (i.e., size, throughput, data protection, privacy, etc.) for the physical storage of the I2RS log file. Data retention policies of the I2RS traceability log is also outside the scope of this document.

### 6.3. Trace Log Rotation

In order to prevent the exhaustion of resources on the I2RS Agent or its associated network device, the I2RS trace log implementation MAY choose to implement a configurable rotation schedule. It SHOULD be possible to do file rotation based on either time or size of the current trace log. If file rollover is supported, multiple archived log files SHOULD be supported in order to maximize the troubleshooting and accounting benefits of the trace log.

### 6.4. Trace Log Retrieval

Implementors are free to provide their own, proprietary interfaces and develop custom tools to retrieve and display the I2RS trace log. These may include the display of the I2RS trace log as Command Line Interface (CLI) output. However, a key intention of defining this information model is to establish an implementor-agnostic and consistent interface to collect I2RS trace data. Correspondingly, retrieval of the data should also be made implementor-agnostic.

The following three sections describe potential ways the trace log can be accessed. At least one of these three MUST be used, with the I2RS mechanisms being preferred as they are implementor-independent approaches to retrieving the data.

#### 6.4.1. Retrieval Via Syslog

The syslog protocol [RFC5424] is a standard way of sending event notification messages from a host to a collector. However, the protocol does not define any standard format for storing the messages, and thus implementors of I2RS tracing would be left to define their own format. So, while the data contained within the syslog message would adhere to this information model, and may be consumable by a human operator, it would not be easily parseable by a machine. Therefore, syslog MAY be employed as a means of retrieving or disseminating the I2RS trace log contents.

#### 6.4.2. Retrieval Via I2RS Information Collection

Section 6.7 of the I2RS architecture [I-D.ietf-i2rs-architecture] defines a mechanism for information collection. The information collected includes obtaining a snapshot of a large amount of data from the network element. It is the intent of I2RS to make this data available in an implementor-agnostic fashion. Therefore, the I2RS trace log SHOULD be made available via the I2RS information collection mechanism either as a single snapshot or via a subscription stream.

#### 6.4.3. Retrieval Via I2RS Pub-Sub

Section 6.7 of the I2RS architecture [I-D.ietf-i2rs-architecture] goes on to define a publish-subscribe mechanism for a feed of changes happening within the I2RS layer. I2RS Agents SHOULD support publishing I2RS trace log information to that feed as described in that document. Subscribers would then receive a live stream of I2RS interactions in trace log format and could flexibly choose to do a number of things with the log messages. For example, the subscribers could log the messages to a datastore, aggregate and summarize interactions from a single client, etc. Using pub-sub for the purpose of logging I2RS interactions augments the areas described by [I-D.camwinget-i2rs-pubsub-sec]. The full range of potential activities is virtually limitless and the details of how they are performed are outside the scope of this document, however.

#### 7. IANA Considerations

This document makes no request of IANA.

#### 8. Security Considerations

The I2RS trace log, like any log file, reveals the state of the entity producing it as well as the identifying information elements and detailed interactions of the system containing it. The information model described in this document does not itself introduce any security issues, but it does define the set of attributes that make up an I2RS log file. These attributes may contain sensitive information and thus should adhere to the security, privacy and permission policies of the organization making use of the I2RS log file.

It is outside the scope of this document to specify how to protect the stored log file, but it is expected that adequate precautions and security best practices such as disk encryption, appropriately restrictive file/directory permissions, suitable hardening and physical security of logging entities, mutual authentication, transport encryption, channel confidentiality, and channel integrity if transferring log files. Additionally, the potentially sensitive information contained in a log file SHOULD be adequately anonymized or obfuscated by operators to ensure its privacy.

#### 9. References

##### 9.1. Normative References

[I-D.ietf-i2rs-architecture]

Atlas, A., Halpern, J., Hares, S., Ward, D., and T. Nadeau, "An Architecture for the Interface to the Routing System", draft-ietf-i2rs-architecture-00 (work in progress), August 2013.

[I-D.ietf-i2rs-problem-statement]

Atlas, A., Nadeau, T., and D. Ward, "Interface to the Routing System Problem Statement", draft-ietf-i2rs-problem-statement-00 (work in progress), August 2013.

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

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.

[RFC4122] Leach, P., Mealling, M., and R. Salz, "A Universally Unique Identifier (UUID) URN Namespace", RFC 4122, July 2005.

[RFC5234] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, January 2008.

## 9.2. Informative References

[I-D.camwinget-i2rs-pubsub-sec]

Beck, K., Cam-Winget, N., and D. McGrew, "Using the Publish-Subscribe Model in the Interface to the Routing System", draft-camwinget-i2rs-pubsub-sec-00 (work in progress), July 2013.

[I-D.nitinb-i2rs-rib-info-model]

Bahadur, N., Folkes, R., Kini, S., and J. Medved, "Routing Information Base Info Model", draft-nitinb-i2rs-rib-info-model-02 (work in progress), August 2013.

[RFC3339] Klyne, G., Ed. and C. Newman, "Date and Time on the Internet: Timestamps", RFC 3339, July 2002.

[RFC5424] Gerhards, R., "The Syslog Protocol", RFC 5424, March 2009.

[RFC5737] Arkko, J., Cotton, M., and L. Vegoda, "IPv4 Address Blocks Reserved for Documentation", RFC 5737, January 2010.

## Authors' Addresses

Joe Clarke  
Cisco Systems, Inc.  
7200-12 Kit Creek Road  
Research Triangle Park, NC 27709  
US

Phone: +1-919-392-2867  
Email: jclarke@cisco.com

Gonzalo Salgueiro  
Cisco Systems, Inc.  
7200-12 Kit Creek Road  
Research Triangle Park, NC 27709  
US

Email: gsalguei@cisco.com

Carlos Pignataro  
Cisco Systems, Inc.  
7200-12 Kit Creek Road  
Research Triangle Park, NC 27709  
US

Email: cpignata@cisco.com



Network Working Group  
Internet-Draft  
Intended status: Informational  
Expires: February 17, 2014

A. Atlas  
Juniper Networks  
J. Halpern  
Ericsson  
S. Hares  
ADARA  
D. Ward  
Cisco Systems  
T. Nadeau  
Juniper Networks  
August 16, 2013

An Architecture for the Interface to the Routing System  
draft-ietf-i2rs-architecture-00

Abstract

This document describes an architecture for a standard, programmatic interface for state transfer in and out of the Internet's routing system. It describes the basic architecture, the components, and their interfaces with particular focus on those to be standardized as part of I2RS.

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 17, 2014.

Copyright Notice

Copyright (c) 2013 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents

(<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

1. Introduction . . . . .	3
1.1. Functional Overview . . . . .	3
1.2. Architectural Overview . . . . .	4
2. Terminology . . . . .	7
3. Key Architectural Properties . . . . .	8
3.1. Simplicity . . . . .	8
3.2. Extensibility . . . . .	9
3.3. Model-Driven Programmatic Interfaces . . . . .	9
3.4. Authorization and Authentication . . . . .	10
4. Network Applications and I2RS Client . . . . .	10
4.1. Example Network Application: Topology Manager . . . . .	11
5. I2RS Agent Role and Functionality . . . . .	11
5.1. Relationship to its Routing Element . . . . .	11
5.2. State Storage . . . . .	12
5.2.1. Starting and Ending . . . . .	12
5.2.2. Reversion . . . . .	13
5.3. Interactions with Local Config . . . . .	13
5.4. Routing Components and Associated I2RS Services . . . . .	13
5.4.1. Unicast and Multicast RIB and LFIB . . . . .	14
5.4.2. IGPs, BGP and Multicast Protocols . . . . .	14
5.4.3. MPLS . . . . .	15
5.4.4. Policy and QoS Mechanisms . . . . .	15
6. I2RS Client Agent Interface . . . . .	15
6.1. Protocol Structure . . . . .	15
6.2. Channel . . . . .	16
6.3. Negotiation . . . . .	16
6.4. Identity and Security Role . . . . .	16
6.4.1. Client Redundancy . . . . .	16
6.5. Connectivity . . . . .	17
6.6. Notifications . . . . .	17
6.7. Information collection . . . . .	18
6.8. Multi-Headed Control . . . . .	18
6.9. Transactions . . . . .	19
7. Manageability Considerations . . . . .	19
8. Security Considerations . . . . .	20
9. IANA Considerations . . . . .	20
10. Acknowledgements . . . . .	20
11. Informative References . . . . .	20

Authors' Addresses . . . . .	20
------------------------------	----

## 1. Introduction

Routers that form the Internet's routing infrastructure maintain state at various layers of detail and function. For example, a typical router maintains a Routing Information Base (RIB), and implements routing protocols such as OSPF, ISIS, BGP to exchange protocol state and other information about the state of the network with other routers.

A router also has information that may be required for applications to understand the network, verify that programmed state is installed in the forwarding plane, measure the behavior of various flows, routes or forwarding entries, as well as understand the configured and active states of the router. Furthermore, routers are typically configured with procedural or policy-based instructions that tell them how to convert all of this information into the forwarding operations that are installed in the forwarding plane. It is also the active state information that describes the expected and observed operational behavior of the router.

This document sets out an architecture for a common, standards-based interface to this information. This Interface to the Routing System (I2RS) facilitates control and diagnosis of the RIB manager's state, as well as enabling network applications to be built on top of today's routed networks. The I2RS is a programmatic asynchronous interface for transferring state into and out of the Internet's routing system, and recognizes that the routing system and a router's OS provide useful mechanisms that applications could harness to accomplish application-level goals.

Fundamental to the I2RS are clear data models that define the semantics of the information that can be written and read. The I2RS provides a framework for registering for and requesting the appropriate information for each particular application. The I2RS provides a way for applications to customize network behavior while leveraging the existing routing system as much as desired.

The I2RS, and therefore this document, are specifically focused on an interface for routing data.

### 1.1. Functional Overview

There are four key aspects to the I2RS. First, the interface is a programmatic interface which needs to be asynchronous and offers fast, interactive access. Second, the I2RS gives access to information and state that is not usually configurable or modeled in

existing implementations or configuration protocols. Third, the I2RS gives applications the ability to learn additional, structured, filterable information and events from the router. Fourth, the I2RS will be data-model driven to facilitate extensibility and provide standard data-models to be used by network applications.

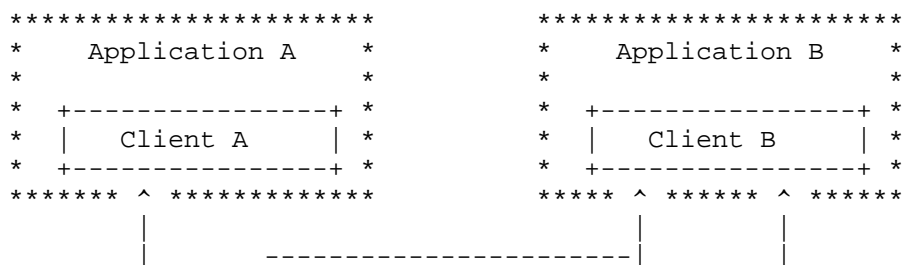
I2RS is described as an asynchronous programmatic interface; the key properties of which are described in Section 5 of [I-D.atlas-i2rs-problem-statement].

Such an interface facilitates the specification of implicitly non-permanent state into the routing system, that can optionally be made permanent. In addition, the extraction of that information and additional dynamic information from the routing system is a critical component of the interface. A non-routing protocol or application could inject state into a routing element via the state-insertion aspects of the I2RS and that state could then be distributed in a routing or signaling protocol and/or be used locally (e.g. to program the co-located forwarding plane).

There are several types of information that the I2RS will facilitate an I2RS Client obtaining. These range from dynamic event notifications (e.g. changes to a particular next-hop, interface up/down, etc.) to information collection streams (statistics, topology, route changes, etc) to simply read operations. The I2RS provides the ability for an I2RS client to request filtered and thresholded information as well as events.

## 1.2. Architectural Overview

The figure in Figure 1 shows the basic architecture for I2RS. Inside a Routing Element, the I2RS agent interacts with both the routing subsystem and with local configuration. A network application uses an I2RS client to communicate with one or more I2RS agents on their routing elements. The scope of I2RS is to define the interactions between the I2RS agent and the I2RS client and the associated proper behavior of the I2RS agent and I2RS client.



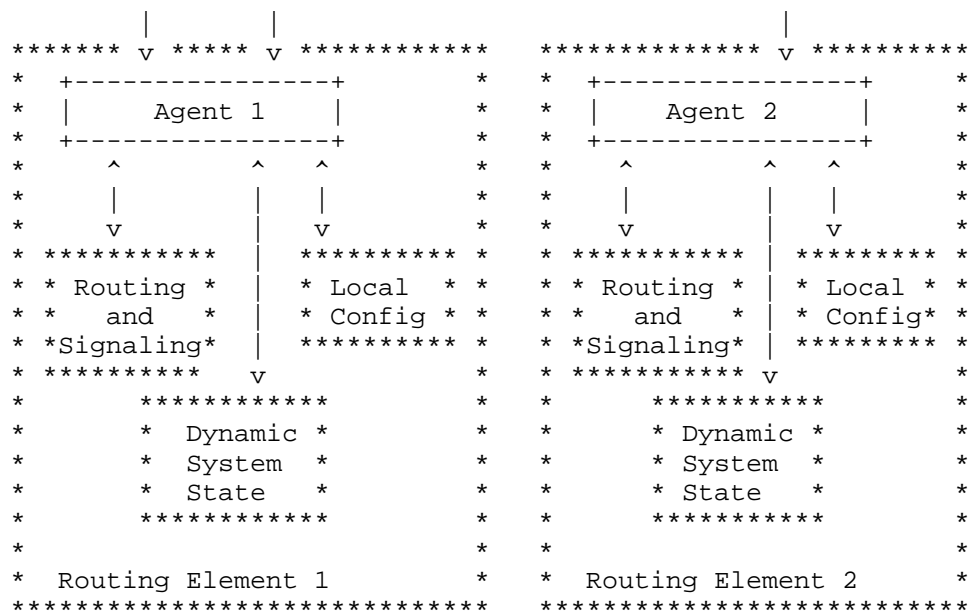


Figure 1: Architecture of I2RS clients and agents

**Routing Element:** A Routing Element implements at least some portion of the routing system. It does not need to have a forwarding plane associated with it. Examples of Routing Elements can include:

A router with a forwarding plane and RIB Manager that runs ISIS, OSPF, BGP, PIM, etc.

A server that runs BGP as a Route Reflector

An LSR that implements RSVP-TE, OSPF-TE, and PCEP and has a forwarding plane and associated RIB Manager.

A server that runs ISIS, OSPF, BGP and uses ForCES to control a remote forwarding plane.

A Routing Element may be locally managed, whether via CLI, SNMP, or NETCONF.

**Routing and Signaling:** This block represents that portion of the Routing Element that implements part of the Internet routing system. It includes not merely standardized protocols (i.e. IS-IS, OSPF, BGP, PIM, RSVP-TE, LDP, etc.), but also the RIB Manager layer.

**Local Config:** A Routing Element will provide the ability to configure and manage it. The Local Config may be provided via a combination of CLI, NETCONF, SNMP, etc. The black box behavior for interactions between the state that I2RS installs into the routing element and the Local Config must be defined.

**Dynamic System State:** An I2RS agent needs access to state on a routing element beyond what is contained in the routing subsystem. Such state may include various counters, statistics, and local events. How this information is provided to the I2RS agent is out of scope, but the standardized information and data models for what is exposed are part of I2RS.

**I2RS Agent:** The I2RS agent implements the I2RS protocol(s) and interacts with the routing element to provide specified behavior.

**Application:** A network application that needs to manipulate the network to achieve its service requirements.

**I2RS Client:** The I2RS client implements the I2RS protocol(s). It interacts with other elements of the policy, provisioning, and configuration system by means outside of the scope of the I2RS effort. It interacts with the I2RS agents to collect information from the routing and forwarding system. Based on the information and the policy oriented interactions, the I2RS client may also interact with the I2RS agent to modify the state of the routing system the client interacts with to achieve operational goals.

As can be seen in Figure 1, an I2RS client can communicate with multiple I2RS agents. An I2RS client may connect to one or more I2RS agents based upon its needs. Similarly, an I2RS agent may communicate with multiple I2RS clients - whether to respond to their requests, to send notifications, etc. Timely notifications are critical so that several simultaneously operating applications have up-to-date information on the state of the network.

As can also be seen in Figure 1, an I2RS Agent may communicate with multiple clients. Each client may send the agent a variety of write operations. The handling of this situation has been a source of discussion in the working group. In order to keep the protocol simple, the current view is that two clients should not be attempting to write (modify) the same piece of information. Such collisions may

happen, but are considered error cases that should be resolved by the network applications and management systems.

Multiple I2RS clients may need to supply data into the same list (e.g. a prefix or filter list); this is not considered an error and must be correctly handled. The nuances so that writers do not normally collide should be handled in the information models.

The architectural goal for the I2RS is that such errors should produce predictable behaviors, and be reportable to interested clients. The details of the associated policy is discussed in Section 6.8. The same policy mechanism (simple priority per I2RS client) applies to interactions between the I2RS agent and the CLI/SNMP/NETCONF as described in Section 5.3.

In addition it must be noted that there may be indirect interactions between write operations. Detection and avoidance of such interactions is outside the scope of the I2RS work and is left to agent design and implementation for now. [[Editor's note: This topic needs more discussion in the working group.]]

## 2. Terminology

The following terminology is used in this document.

**agent or I2RS Agent:** An I2RS agent provides the supported I2RS services to the local system's routing sub-systems. The I2RS agent understands the I2RS protocol and can be contacted by I2RS clients.

**client or I2RS Client:** A client speaks the I2RS protocol to communicate with I2RS Agents and uses the I2RS services to accomplish a task. An I2RS client can be seen as the part of an application that uses and supports I2RS and could be a software library.

**service or I2RS Service:** For the purposes of I2RS, a service refers to a set of related state access functions together with the policies that control their usage. The expectation is that a service will be represented by a data-model. For instance, 'RIB service' could be an example of a service that gives access to state held in a device's RIB.

**read scope:** The set of information which the I2RS client is authorized to read. This access includes the permission to see the existence of data and the ability to retrieve the value of that data.

notification scope: The set of events and associated information that the I2RS Client can request be pushed by the I2RS Agent. I2RS Clients have the ability to register for specific events and information streams, but must do so given the access restrictions of their notification scope.

write scope: The set of field values which the I2RS client is authorized to write (i.e. add, modify or delete). This access can restrict what data can be modified or created, and what specific value sets and ranges can be installed.

scope: When unspecified as either read scope or write scope, the term scope applies to both the read scope and write scope.

resources: A resource is an I2RS-specific use of memory, storage, or execution that a client may consume due to its I2RS operations. The amount of each such resource that a client may consume in the context of a particular agent can be constrained based upon the client's security role. An example of such a resource could include the number of notifications registered for. These are not protocol-specific resources or network-specific resources.

role or security role: A security role specifies the scope, resources, priorities, etc. that a client or agent has.

identity: A client is associated with exactly one specific identity. State can be attributed to a particular identity. It is possible for multiple communication channels to use the same identity; in that case, the assumption is that the associated client is coordinating such communication.

secondary identity: An I2RS Client may supply a secondary opaque identity that is not interpreted by the I2RS Agent. An example use is when the I2RS Client is a go-between for multiple applications and it is necessary to track which application has requested a particular operation.

### 3. Key Architectural Properties

#### 3.1. Simplicity



There have been many efforts over the years to improve the access to the information known to the routing and forwarding system. Making such information visible and usable to network management and applications has many well-understood benefits. There are two related challenges in doing so. First, the span of information potentially available is very large. Second, the variation both in the structure of the data and in the kinds of operations required tends to introduce protocol complexity.

Having noted that, it is also critical to the utility of I2RS that it be easily deployed and robust. Complexity in the protocol hinders implementation, robustness, and deployability. Also, complexity in the data models frequently makes it harder to extend rather than easier.

Thus, one of the key aims for I2RS is the keep the protocol and modeling architecture simple. So for each architectural component or aspect, we ask ourselves "do we need this complexity, or is the behavior merely nice to have?" Protocol parsimony is clearly a goal.

### 3.2. Extensibility

There are several ways that the scope of the I2RS work is being restricted in the interests of achieving a deliverable and deployable result. We are only working on the models to be used over the single identified interface. We are only looking at modeling a subset of the data of interest. And we are probably only representing a subset of the operations that may eventually be needed (although there is some hope that we are closer on that aspect than others to what is needed.) Thus, it is important to consider extensibility not only of the underlying services' data models, but also of the primitives and protocol operations.

At the same time, it is clearly desirable for the data models and protocol operations we define in the I2RS to be useful in more general settings. It should be easy to integrate data models from the I2RS with other data. Other work should be able to easily extend it to represent additional aspects of the network elements or network systems. Hence, the data model and protocol definitions need to be designed to be highly extensible, preferably in a regular and simple fashion.

### 3.3. Model-Driven Programmatic Interfaces

A critical component of I2RS is the standard information and data models with their associated semantics. While many components of the routing system are standardized, associated data models for them are not yet available. Instead, each router uses different information,

different mechanisms, and different CLI which makes a standard interface for use by applications extremely cumbersome to develop and maintain. Well-known data modeling languages exist and may be used for defining the data models for I2RS.

There are several key benefits for I2RS in using model-driven architecture and protocol(s). First, it allows for transferring data-models whose content is not explicitly implemented or understood. Second, tools can automate checking and manipulating data; this is particularly valuable for both extensibility and for the ability to easily manipulate and check proprietary data-models.

The different services provided by I2RS can correspond to separate data-models. An I2RS agent may indicate which data-models are supported.

### 3.4. Authorization and Authentication

All control exchanges between the I2RS client and agent MUST be authenticated and integrity protected (such that the contents cannot be changed without detection). Manipulation of the system must be accurately attributable. In an ideal architecture, even information collection and notification should be protected; this may be subject to engineering tradeoffs during the design.

I2RS Agents, in performing information collection and manipulation, will be acting on behalf of the I2RS clients. As such, they will operate based on the lower of the two permissions of the agent itself and of the client.

I2RS clients may be operating on behalf of other applications. While those applications' identities are not need for authorization, each application should have a unique opaque identifier that can be provided by the I2RS client to the I2RS agent for purposes of tracking attribution of operations to support functionality such as accounting and troubleshooting.

## 4. Network Applications and I2RS Client

An I2RS Client has a standardized interface that uses the I2RS protocol(s) to communicate with I2RS Agents. The interface between an I2RS client and the network applications is outside the scope of I2RS.

When an I2RS Client interacts with multiple network applications, that I2RS Client is behaving as a go-between and should indicate this to the I2RS Agents by, for example, specifying a secondary opaque identity to allow improved troubleshooting.

A network application that uses an I2RS client may also be considered a routing element and include an I2RS agent for interactions. However, where the needed information and data models for that upper interface differs from that of a conventional routing element, those models are, at least initially, out of scope for I2RS.

#### 4.1. Example Network Application: Topology Manager

One example of such an application is a Topology Manager. A Topology Manager includes an I2RS client that uses the I2RS data models and protocol to collect information about the state of the network by communicating directly with one or more I2RS agents. From these I2RS agents, the Topology Manager collects routing configuration and operational data. Most importantly, it collects information about the routing system, including the contents of the IGP (e.g., IS-IS or OSPF) and BGP data sets.

The Topology Manager may be embedded as a component of a larger application. It would construct internal data structures and use the collected data to drive functions such as path computations or anomalous routing detection. Alternatively, the Topology Manager could combine the I2RS-collected data with other information, abstract a composite set, and provide a coherent picture of the network state accessible via another interface. That interface might use the same I2RS protocol and could use extensions to the I2RS data models. Developing such mechanisms is outside the initial scope of the I2RS work.

### 5. I2RS Agent Role and Functionality

The I2RS Agent is part of a routing element. As such, it has relationships with that routing element as a whole, and with various components of that routing element.

#### 5.1. Relationship to its Routing Element

A Routing Element may be implemented with a wide variety of different architectures: an integrated router, a split architecture, distributed architecture, etc. The architecture does not need to affect the general I2RS agent behavior.

For scalability and generality, the I2RS agent may be responsible for collecting and delivering large amounts of data from various parts of the routing element. Those parts may or may not actually be part of a single physical device. Thus, for scalability and robustness, it is important that the architecture allow for a distributed set of reporting components providing collected data from the I2RS agent back to the relevant I2RS clients. As currently envisioned, a given

I2RS agent would have only one locus per I2RS service for manipulation of routing element state.

## 5.2. State Storage

State modification requests are sent to the I2RS agent in a network element by I2RS clients. The I2RS agent is responsible for applying these changes to the system. How much data must the I2RS Agent store about these state-modifying operations, and with what persistence? There are range of possible answers. One extreme is where it stores nothing, cannot indicate why or by whom state was placed into the routing element, and relies on clients reapplying things in all possible cases. The other extreme is where multiple clients' overlapping operations are stored and managed, as is done in the RIB for routes with a preference or priority to pick between the routes.

In answering this question, this architecture tries to provide sufficient power to keep client operations effective, while still being simple to implement in the I2RS Agent, and to observe meaningfully during operation. The I2RS agent stores the set of operations it has applied. Simply, the I2RS agent stores who did what operation to which entity. New changes replace any data about old ones. If an I2RS client does an operation to remove some state, that state is removed and the I2RS agent stores no more information about it. This allows any interested party to determine what the current effect of I2RS on the system is, and why. Meaningful logging is also recommended.

The I2RS Agent will not attempt to retain or reapply state across routing element reboot. Determination of whether state still applies depends heavily on the causes of reboots, and reapplication is at least as likely to cause problems as it is to provide for correct operation. [[Editor's note: This topics needs more discussion in the working group.]]

### 5.2.1. Starting and Ending

An I2RS client applies changes via the I2RS protocol based on policy and other application inputs. While these changes may be of the form "do this now, and leave it there forever", they are frequently driven by other conditions which may have start times, stop times, or are only to be used under certain conditions. The I2RS interface protocol could be designed to allow an I2RS Client to provide a wide range of such conditional information to the I2RS Agent for application. At the other extreme, the I2RS client could provide all such functionality based on its own clocking and network event reporting from the relevant I2RS Agents.

Given that the complexity of possible conditions is very large, and that some conditions may even cross network element boundaries, clearly some degree of handling must be provided on the I2RS client. As such, in this architecture it is assumed that all the complexity associated with this should be left to the I2RS client. This architectural view does mean that reliability of the communication path between the I2RS client and I2RS agent is critical. [[Editor's note: This requires more discussion in the working group.]]

#### 5.2.2. Reversion

An I2RS Agent may decide that some state should no longer be applied. An I2RS Client may instruct an Agent to remove state it has applied. In all such cases, the state will revert to what it would have been without the I2RS; that state is generally whatever was specified via the CLI, NETCONF, SNMP, etc. I2RS Agents will not store multiple alternative states, nor try to determine which one among such a plurality it should fall back to. Thus, the model followed is not like the RIB, where multiple routes are stored at different preferences.

An I2RS Client may register for notifications when state that was applied by a particular I2RS Client is modified or removed.

#### 5.3. Interactions with Local Config

As described above, local device configuration is considered to be separate from the I2RS data store. Thus, changes may originate from either source. Policy (i.e. comparisons between a CLI/SNMP/NETCONF priority and a I2RS agent priority) can determine whether the local configuration should overwrite any state written by I2RS and attributed to a particular I2RS Client or whether I2RS as attributed to a particular I2RS Client can overwrite local configuration state.

Simply allowing the most recent state to prevail could cause race conditions where the final state is not repeatably deterministic. One important aspect is that if CLI/SNMP/NETCONF changes data that is subject to monitoring or manipulating by I2RS, then the system must be instrumented enough to provide suitable I2RS notifications of these changes.

#### 5.4. Routing Components and Associated I2RS Services

For simplicity, each logical protocol or set of functionality that be compactly described in a separable information and data model is considered as a separate I2RS Service. A routing element need not implement all routing components described nor provide the associated I2RS services. The initial services included in the I2RS architecture are as follows.

#### 5.4.1. Unicast and Multicast RIB and LFIB

Network elements concerned with routing IP maintain IP unicast RIBs. Similarly, there are RIBs for IP Multicast, and a Label Information Base (LIB) for MPLS. The I2RS Agent needs to be able to read and write these sets of data. The I2RS data model must include models for this information.

In particular, with regard to writing this information, the I2RS Agent should use the same mechanisms that the routing element already uses to handle RIB input from multiple sources, so as to compatibly change the system state.

The multicast state added to the multicast RIB does not need to match to well-known protocol installed state. The I2RS Agent can create arbitrary replication state in the RIB, subject to the advertised capabilities of the routing element.

#### 5.4.2. IGPs, BGP and Multicast Protocols

In addition to interacting with the consolidated RIB, the I2RS agent may need to interact with the individual routing protocols on the device. This interaction includes a number of different kinds of operations:

- o reading the various internal rib(s) of the routing protocol is often helpful for understanding the state of the network. Directly writing these protocol-specific RIBs or databases is out of scope for I2RS.
- o reading the various pieces of policy information the particular protocol instance is using to drive its operations.
- o writing policy information such as interface attributes that are specific to the routing protocol or BGP policy that may indirectly manipulate attributes of routes carried in BGP.
- o writing routes or prefixes to be advertised via the protocol.
- o joining/removing interfaces from the multicast trees

- o subscribing to an information stream of route changes
- o receiving notifications about peers coming up or going down

For example, the interaction with OSPF might include modifying the local routing element's link metrics, announcing a locally-attached prefix, or reading some of the OSPF link-state database. However, direct modification of the link-state database is NOT allowed to preserve network state consistency.

#### 5.4.3. MPLS

The I2RS agent will need to interact with the protocols that create transport LSPs (e.g. LDP and RSVP-TE) as well as protocols (e.g. BGP, LDP) that provide MPLS-based services (e.g. pseudowires, L3VPNs, L2VPNs, etc).

#### 5.4.4. Policy and QoS Mechanisms

Many network elements have separate policy and QoS mechanisms, including knobs which affect local path computation and queue control capabilities. These capabilities vary widely across implementations, and I2RS cannot model the full range of information collection or manipulation of these attributes. A core set does need to be included in the I2RS data models and in the expected interfaces between the I2RS Agent and the network element, in order to provide basic capabilities and the hooks for future extensibility.  
[[Editor's note: This requires more discussion in the working group.]]

### 6. I2RS Client Agent Interface

#### 6.1. Protocol Structure

One could view I2RS merely as a way to talk about the existing network management interfaces to a network element. That would be quite limiting and would not meet the requirements elucidated elsewhere. One could also view I2RS as a collection of protocols - some existing and some new - that meet the needs. While that could be made to work, the complexity of such a mechanism would be quite high. One would need to develop means to coordinate information across a set of protocols that were not designed to work together. From a deployability perspective, this would not meet the goal of simplicity. As a result, this architecture views the I2RS as an interface supporting a single control and data exchange protocol. Whether such a protocol is built upon extending existing mechanisms or requires a new mechanism requires further investigation. That protocol may use several underlying transports (TCP, SCTP, DCCP),

with suitable authentication and integrity protection mechanisms. These different transports can support different types of communication (e.g. control, reading, notifications, and information collection) and different sets of data. Whatever transport is used for the data exchange, it must also support suitable congestion control mechanisms.

## 6.2. Channel

The uses of a single I2RS protocol does not imply that only one channel of communication is required. There may be a range of reliability requirements, and to support the scaling there may need to be channels originating from multiple sub-components of a routing element. These will all use the data exchange protocol, and establishment of additional channels for communication will be coordinated between the I2RS client and the I2RS agent.

## 6.3. Negotiation

Protocol support capabilities will vary across I2RS Clients and Routing Elements supporting I2RS Agents. As such, capability negotiation (such as which transports are supported beyond the minimum required to implement) will clearly be necessary. It is important that such negotiations be kept simple and robust, as such mechanisms are often a source of difficulty in implementation and deployment.

Negotiation should be broken into several aspects, such as protocol capabilities and I2RS services and model types supported.

## 6.4. Identity and Security Role

Each I2RS Client will have a unique identity; it can also have secondary identities to be used for troubleshooting. A secondary identity is merely a unique, opaque identifier that may be helpful in troubleshooting. Via authentication and authorization mechanisms, the I2RS agent will have a specific scope for reading data, for writing data, and limitations on the resources that can be consumed. The scopes need to specify both the data and the value ranges.

### 6.4.1. Client Redundancy

I2RS must support client redundancy. At the simplest, this can be handled by having a primary and a backup network application that both use the same client identity and can successfully authenticate as such. Since I2RS does not require a continuous transport connection and supports multiple transport sessions, this can provide some basic redundancy. However, it does not address concerns for



troubleshooting and accountability about knowing which network application is actually active. At a minimum, basic transport information about each connection and time can be logged with the identity. Further discussion is necessary to determine whether additional client identification information is necessary. [[Editor's note: This requires more discussion in the working group.]]

#### 6.5. Connectivity

A client may or may not maintain an active communication channel with an agent. Therefore, an agent may need to open a communication channel to the client to communicate previously requested information. The lack of an active communication channel does not imply that the associated client is non-functional. When communication is required, the agent or client can open a new communication channel.

State held by an agent that is owned by a client should not be removed or cleaned up when a client is no longer communicating - even if the agent cannot successfully open a new communication channel to the client.

There are three different assumptions that can apply to handling dead clients. The first is that the network applications or management systems will detect a dead network application and either restart that network application or clean up any state left behind. The second is to allow state expiration, expressed as a policy associated with the I2RS client's role. The state expiration could occur after there has been no successful communication channel to or from the I2RS client for the policy-specified duration. The third is that the client could explicitly request state clean-up if a particular transport session is terminated.

#### 6.6. Notifications

As with any policy system interacting with the network, the I2RS Agent needs to be able to receive notifications of changes in network state. Notifications here refers to changes which are unanticipated, represent events outside the control of the systems (such as interface failures on controlled devices), or are sufficiently sparse as to be anomalous in some fashion.

Such events may be of interest to multiple I2RS Clients controlling data handled by an I2RS Agent, and to multiple other I2RS clients which are collecting information without exerting control. The architecture therefore requires that it be practical for I2RS Clients to register for a range of notifications, and for the I2R Agents to send notifications to a number of Clients.

As the I2RS is developed, it is likely that a management information-model and data-model will be required to describe event notifications for general or I2RS errors.

For performance and scaling by the I2RS client and general information privacy, an I2RS Client needs to be able to register for just the events it is interested in. It is also possible that I2RS might provide a stream of notifications via a publish/subscribe mechanism that is not amenable to having the I2RS agent do the filtering.

#### 6.7. Information collection

One of the other important aspects of the I2RS is that it is intended to simplify collecting information about the state of network elements. This includes both getting a snapshot of a large amount of data about the current state of the network element, and subscribing to a feed of the ongoing changes to the set of data or a subset thereof. This is considered architecturally separate from notifications due to the differences in information rate and total volume.

#### 6.8. Multi-Headed Control

As was described earlier, an I2RS Agent interacts with multiple I2RS Clients who are actively controlling the network element. From an architecture and design perspective, the assumption is that by means outside of this system the data to be manipulated within the network element is appropriately partitioned so that any given piece of information is only being manipulated by a single I2RS Client.

Nonetheless, unexpected interactions happen and two (or more) I2RS clients may attempt to manipulate the same piece of data. This is considered an error case. This architecture does not attempt to determine what the right state of data is in such a collision. Rather, the architecture mandates that there be decidable means by which I2RS Agents will handle the collisions. The current recommendation is to have a simple priority associated with each I2RS clients, and the highest priority change remains in effect. In the case of priority ties, the first client whose attribution is associated with the data will keep control.

In order for this to be useful for I2RS Clients, it is important that it be possible for an I2RS Client to register for changes to any I2RS manipulatable data that it may care about. The I2RS client may then respond to the situation as it sees fit.

## 6.9. Transactions

In the interest of simplicity, the I2RS architecture does not include multi-message atomicity and rollback mechanisms. Rather, it includes a small range of error handling for a set of operations included in a single message. An I2RS Client may indicate one of the following three error handling for a given message with multiple operations which it sends to an I2RS Agent:

**Perform all or none:** This traditional SNMP semantic indicates that other I2RS agent will keep enough state when handling a single message to roll back the operations within that message. Either all the operations will succeed, or none of them will be applied and an error message will report the single failure which caused the not to be applied. This is useful when there are, for example, mutual dependencies across operations in the message.

**Perform until error:** In this case, the operations in the message are applied in the specified order. When an error occurs, no further operations are applied, and an error is returned indicating the failure. This is useful if there are dependencies among the operations and they can be topologically sorted.

**Perform all storing errors:** In this case, the I2RS Agent will attempt to perform all the operations in the message, and will return error indications for each one that fails. This is useful when there is no dependency across the operation, or where the client would prefer to sort out the effect of errors on its own.

In the interest of robustness and clarity of protocol state, the protocol will include an explicit reply to modification operations even when they fully succeed.

## 7. Manageability Considerations

Manageability plays a key aspect in I2RS. Some initial examples include:

**Resource Limitations:** Using I2RS, applications can consume resources, whether those be operations in a time-frame, entries in the RIB, stored operations to be triggered, etc. The ability to set resource limits based upon authorization is important.

**Configuration Interactions:** The interaction of state installed via the I2RS and via a router's configuration needs to be clearly defined. As described in this architecture, a simple priority that is configured can be used to express the desired policy.

## 8. Security Considerations

This framework describes interfaces that clearly require serious consideration of security. The ability to identify, authenticate and authorize applications that wish to install state is necessary and briefly described in Section 3.4. Security of communications from the applications is also required as discussed in Section 6.1. Scopes for reading and writing data specified in the context of the data models and the value ranges are discussed briefly in Section 6.4.

## 9. IANA Considerations

This document includes no request to IANA.

## 10. Acknowledgements

Significant portions of this draft came from draft-ward-i2rs-framework-00 and draft-atlas-i2rs-policy-framework-00.

The authors would like to thank Nitin Bahadur, Shane Amante, Ed Crabbe, Ken Gray, Carlos Pignataro, Wes George, Joe Clarke, Juergen Schoenwalder, and Jamal Hadi Salim for their suggestions and review.

## 11. 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-01 (work in progress), July 2013.

### Authors' Addresses

Alia Atlas  
Juniper Networks  
10 Technology Park Drive  
Westford, MA 01886  
USA

Email: akatlas@juniper.net

Joel Halpern  
Ericsson

Email: Joel.Halpern@ericsson.com

Susan Hares  
ADARA

Email: [shares@ndzh.com](mailto:shares@ndzh.com)

Dave Ward  
Cisco Systems  
Tasman Drive  
San Jose, CA 95134  
USA

Email: [wardd@cisco.com](mailto:wardd@cisco.com)

Thomas D. Nadeau  
Juniper Networks

Email: [tnadeau@juniper.net](mailto:tnadeau@juniper.net)

Network Working Group  
Internet-Draft  
Intended status: Informational  
Expires: April 21, 2014

N. Bahadur, Ed.  
  
R. Folkes, Ed.  
Juniper Networks, Inc.  
S. Kini  
Ericsson  
J. Medved  
Cisco  
October 18, 2013

Routing Information Base Info Model  
draft-ietf-i2rs-rib-info-model-01

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 April 21, 2014.

Copyright Notice

Copyright (c) 2013 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

1. Introduction . . . . .	4
1.1. Conventions used in this document . . . . .	6
2. RIB data . . . . .	6
2.1. RIB definition . . . . .	6
2.2. Routing instance . . . . .	7
2.3. Route . . . . .	8
2.4. Nexthop . . . . .	9
2.4.1. Nexthop types . . . . .	12
2.4.2. Nexthop list attributes . . . . .	12
2.4.3. Nexthop content . . . . .	13
2.4.4. Special nexthops . . . . .	14
3. Reading from the RIB . . . . .	14
4. Writing to the RIB . . . . .	15
5. Events and Notifications . . . . .	15
6. RIB grammar . . . . .	16
7. Inter-domain extensions to the RIB . . . . .	18
7.1. Extension to Routing Instance . . . . .	18
7.2. Extension to Route . . . . .	19
7.3. Inter-domain extensions to RIB grammar . . . . .	19
8. Using the RIB grammar . . . . .	19
8.1. Using route preference . . . . .	19
8.2. Using different nexthops types . . . . .	20
8.2.1. Tunnel nexthops . . . . .	20
8.2.2. Replication lists . . . . .	20
8.2.3. Weighted lists . . . . .	20
8.2.4. Protection lists . . . . .	21
8.2.5. Nexthop chains . . . . .	21
8.2.6. Lists of lists . . . . .	22
8.3. Performing multicast . . . . .	22
8.4. Solving optimized exit control . . . . .	22
9. RIB operations at scale . . . . .	23
9.1. RIB reads . . . . .	23
9.2. RIB writes . . . . .	23
9.3. RIB events and notifications . . . . .	23
10. Security Considerations . . . . .	24
11. IANA Considerations . . . . .	24
12. Acknowledgements . . . . .	24
13. References . . . . .	24
13.1. Normative References . . . . .	24
13.2. Informative References . . . . .	24
Authors' Addresses . . . . .	25



## 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.

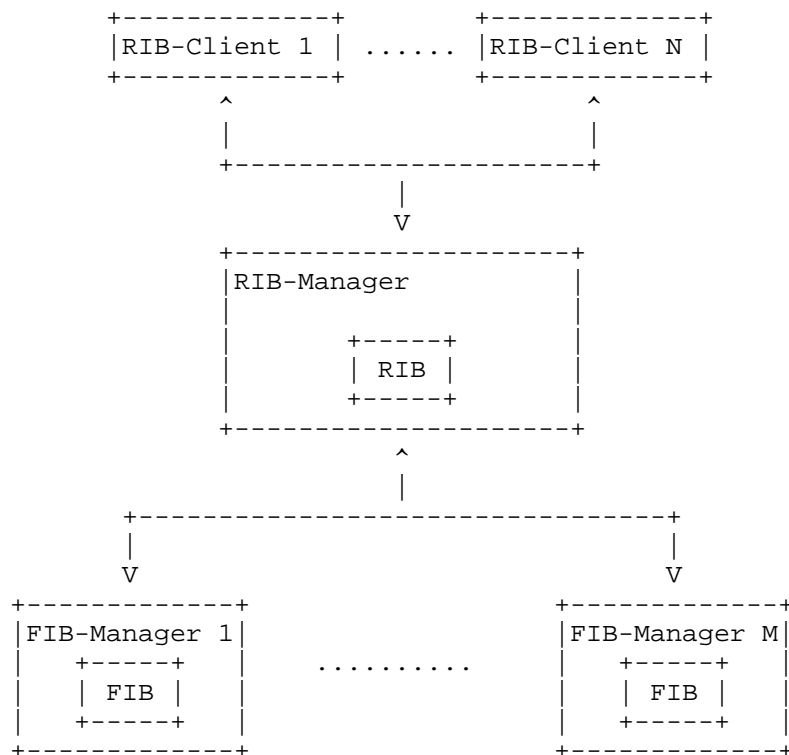


Figure 1: RIB-Manager, RIB-Clients and FIB-Managers

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. Section 7 extends the RIB for use in inter-domain cases. Examples of using the RIB grammar are shown in Section 8. Section 9 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.

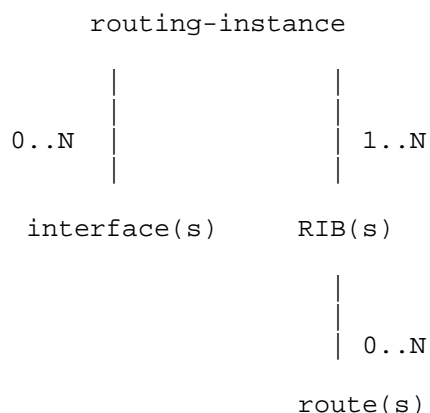


Figure 2: RIB model

### 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.

- o 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.
- o 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.

- o 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.
- o 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.

### 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.

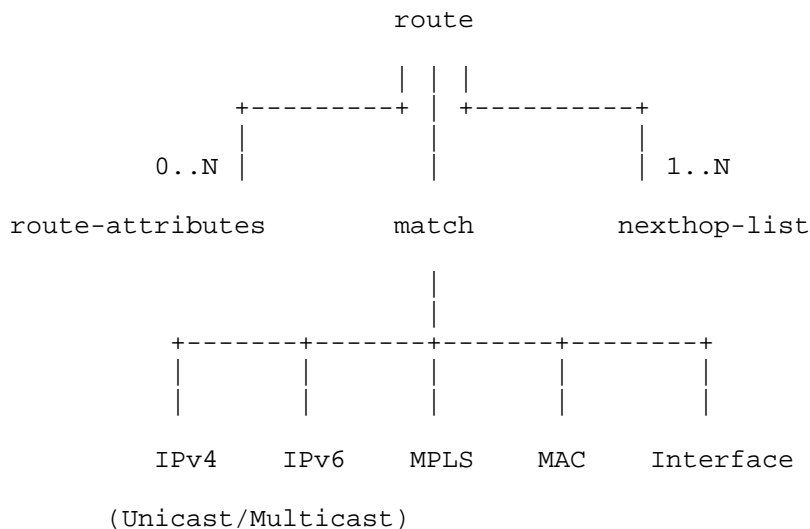


Figure 3: Route model

This document specifies the following match types:

- o IPv4: Match on destination IP in IPv4 header
- o IPv6: Match on destination IP in IPv6 header
- o MPLS: Match on a MPLS tag

- o MAC: Match on ethernet destination addresses
- o Interface: Match on incoming interface of packet
- o 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.

- o 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 8.1.
- o 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.
- o 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.
- o 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.

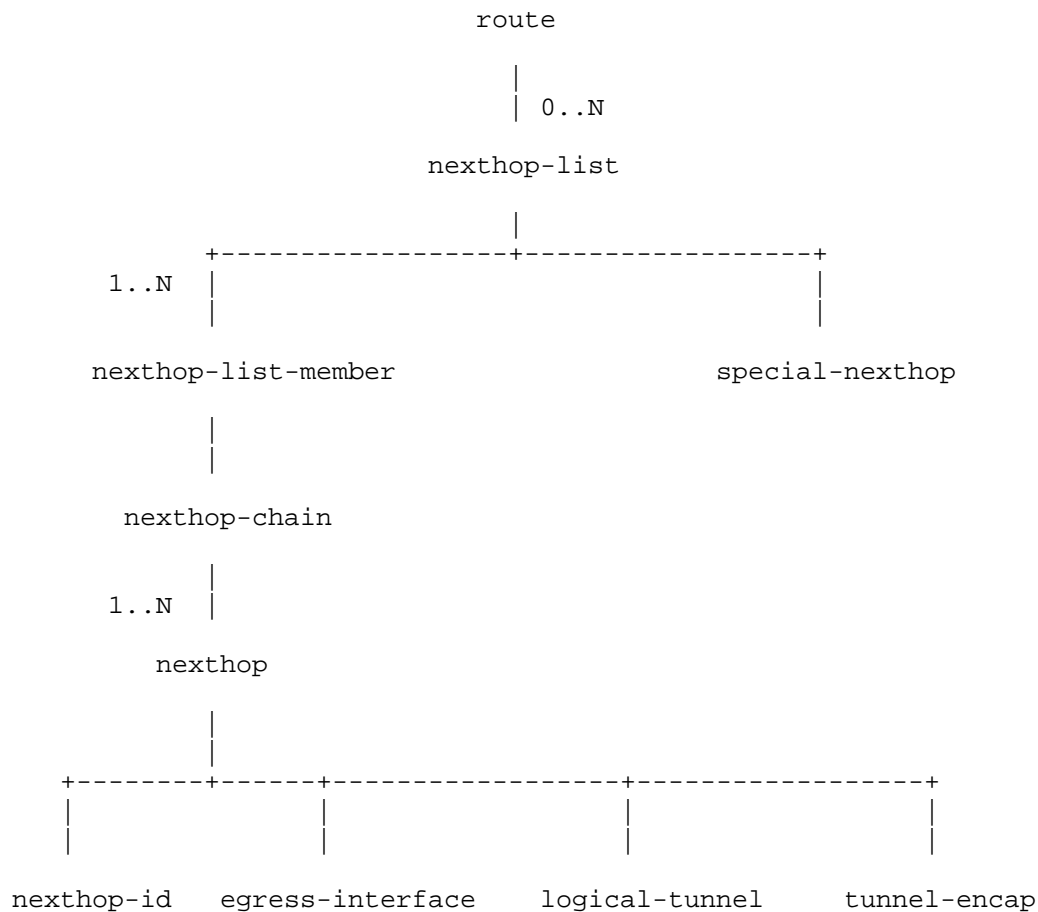


Figure 4: Nexthop model

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

- o Unicast nexthops - pointing to an interface
- o Tunnel nexthops - pointing to a tunnel
- o Replication lists - list of nexthops to which to replicate a packet to
- o Weighted lists - for load-balancing
- o Protection lists - for primary/backup paths
- o Nexthop chains - for chaining headers, e.g. MPLS label over a GRE header
- o Lists of lists - recursive application of the above
- o Indirect nexthops - pointing to a nexthop identifier
- o 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 8.2.

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:

- o 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.
- o 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:

- o identifier: This is an identifier returned by the network device representing another nexthop or another nexthop chain.
- o EGRESS\_INTERFACE: This represents a physical, logical or virtual interface on the network device.
- o address: This can be an IP address or MAC 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.
- o 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.
- o 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).
- o 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. Special nexthops

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

- o DISCARD: This indicates that the network device should drop the packet and increment a drop counter.
- o 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).
- o 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:

- o Installed - Yes/No (Indicates whether the route got installed in the FIB)
- o Active - Yes/No (Indicates whether a route is fully resolved and is a candidate for selection)
- o 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:

- o Route change notification, with return code as specified in Section 4
- o 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>
                        [<interface-list>] <rib-list>
                        [<ROUTER_ID>]

<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> ::= <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_ADDRESS_LENGTH>

<ipv6-route> ::= <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>

<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>] [<LOCAL_ONLY>]

```

```

[<address-family-route-attributes>]

<address-family-route-attributes> ::= <ip-route-attributes> |
                                         <mpls-route-attributes> |
                                         <ethernet-route-attributes>
<ip-route-attributes> ::= [<rpf-check-interface>]
<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-identifier> ::= <NEXTHOP_NAME> | <NEXTHOP_ID>
<nexthop-address> ::= (<IPv4> <ipv4-address>) |
                      (<IPv6> <ipv6-address>) |
                      (<IEEE_MAC> <IEEE_MAC_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>]

```

Figure 5: RIB rBNF grammar

## 7. Inter-domain extensions to the RIB

This section describes inter-domain extensions to the Routing Information Base to allow an external entity to learn about and program inter-domain information associated with the RIB and it's routes. In the absence of this extension, the network device MUST NOT return any routes that have inter-domain information associated (such as autonomous-system path information).

### 7.1. Extension to Routing Instance

A routing instance is augmented with an optional parameter called as-data.

as-data 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]).

## 7.2. Extension to Route

Routes are augmented with an optional parameter called 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].

## 7.3. Inter-domain extensions to RIB grammar

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

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

<ip-route-attributes> ::= [<rpf-check-interface>] [<as-path>]
<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>]
```

Figure 6: RIB rBNF grammar - Inter-domain extensions

## 8. 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.

### 8.1. Using route preference

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.

## 8.2. Using different nexthops types

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

### 8.2.1. 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.

### 8.2.2. 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> ... ]
```

### 8.2.3. 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>) ... ]
<nexthop-list> ::= (<nexthop> <LOAD_BALANCE_WEIGHT>)
                  [(<nexthop> <LOAD_BALANCE_WEIGHT>)... ]
```

#### 8.2.4. 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>)... ]
```

#### 8.2.5. 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>)
```

#### 8.2.6. 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.

#### 8.3. 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 8.2.2 ).

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 8.2.2 ) of <tunnel-encap>, with each tunnel encap representing a single mpls downstream nexthop.

#### 8.4. 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.

## 9. 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.

### 9.1. 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.

### 9.2. 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.

### 9.3. 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.

## 10. 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.

## 11. IANA Considerations

This document does not generate any considerations for IANA.

## 12. 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.

## 13. References

### 13.1. Normative References

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

### 13.2. 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 A. Retana, "Protocol Independent Use Cases for an Interface to the Routing System", draft-white-i2rs-use-case-01 (work in progress), August 2013.

[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-IS)", 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)

Email: [nitin\\_bahadur@yahoo.com](mailto:nitin_bahadur@yahoo.com)

Ron Folkes (editor)

Juniper Networks, Inc.  
1194 N. Mathilda Avenue  
Sunnyvale, CA 94089  
US

Phone: +1 408 745 2000  
Email: [ronf@juniper.net](mailto:ronf@juniper.net)  
URI: [www.juniper.net](http://www.juniper.net)

Sriganesh Kini  
Ericsson

Email: [sriganesh.kini@ericsson.com](mailto:sriganesh.kini@ericsson.com)

Jan Medved  
Cisco

Email: [jmedved@cisco.com](mailto:jmedved@cisco.com)





I2RS Working Group  
Internet Draft  
Category: Informational

R. Krishnan  
Brocade Communications  
A. Ghanwani  
Dell  
S. Kini  
Ericsson  
D. Mcdysan  
Verizon

Expires: April 2014

October 13, 2013

## I2RS Large Flow Use Case

draft-krishnan-i2rs-large-flow-use-case-00

### Abstract

Demands on networking bandwidth are growing exponentially due to applications such as large file transfers and those with rich media. Link Aggregation Group (LAG) and Equal Cost Multipath (ECMP) are extensively deployed in networks to scale the bandwidth. However, the flow based load balancing techniques used today make inefficient use of the bandwidth in the presence of long lived large flows. This draft presents a use-case to improve the efficiency under such conditions and aims to drive requirements for the I2RS WG.

### Status of this Memo

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

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

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."

The list of current Internet-Drafts can be accessed at <http://www.ietf.org/ietf/lid-abstracts.txt>.

The list of Internet-Draft Shadow Directories can be accessed at <http://www.ietf.org/shadow.html>.

This Internet-Draft will expire on April, 2014.

#### Copyright Notice

Copyright (c) 2013 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document.

#### 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 RFC-2119 [RFC 2119].

#### Table of Contents

1. Introduction.....	3
1.1. Acronyms.....	4
1.2. Terminology.....	4
2. Large Flow Recognition and Signaling.....	5
2.1. Network-based Recognition of Large Flows.....	5
2.2. Application-based Signaling of Large Flows.....	5
3. Flow Rebalancing.....	5
3.1. Local Rebalancing.....	5
3.2. Global Rebalancing.....	6
3.2.1. IP Networks.....	6
3.2.2. MPLS Networks.....	7
4. Operational Considerations.....	7
5. IANA Considerations.....	7
6. Security Considerations.....	8
7. Acknowledgements.....	8
8. References.....	8
8.1. Normative References.....	8
8.2. Informative References.....	8
Authors' Addresses.....	8

## 1. Introduction

Networks extensively deploy LAG and ECMP for bandwidth scaling. Network traffic can be predominantly categorized into two traffic types: long-lived large flows and other flows (which include long-lived small flows, short-lived small/large flows) [OPSAWG-large-flow]. Stateless hash-based techniques [ITCOM, RFC 2991, RFC 2992, and RFC 6790] are often used to distribute both long-lived large flows and other flows over the components in a LAG/ECMP. However the traffic may not be evenly distributed over the component links due to the traffic pattern.

This draft describes long-lived large flow load balancing techniques for achieving the best network bandwidth utilization with LAG/ECMP and the corresponding I2RS requirements. Some of these techniques have been described in detail in [OPSAWG-large-flow]. We describe methods that can be used locally within a single router, as well as methods that can be applied across multiple network elements, where the network is under the control of single administrative entity. We refer to the former as local load balancing and the latter as global load balancing. A combination of local and global load balancing helps in achieving the best network bandwidth utilization and latency for a given network topology.

From a router standpoint, long-lived large flows are typically identified using one or more fields from the packet header from the following list:

- . Layer 2: source MAC address, destination MAC address, VLAN ID.
- . IP header: IP Protocol, IP source address, IP destination address, flow label (IPv6 only), TCP/UDP source port, TCP/UDP destination port.
- . MPLS Labels.

For tunneling protocols like GRE, VXLAN, NVGRE, STT, etc., flow identification is possible based on inner and/or outer headers. The above list is not exhaustive.

In the remainder of this document, consistent with [OPSAWG-large-flow], we use the term "large flow" to refer to "long-lived large flows," and we use the term "small flow" to refer to any of the three other types of flows identified above.

At a high-level, the technique involves recognizing large flows and rebalancing them to achieve optimal load balancing. Large flows may be recognized within a router, or using the aid of an external management entity such as an IPFIX [RFC 7011] collector or a sFlow [sFlow-v5] collector. Once a large flow has been recognized, it must be signaled to a management entity that makes the rebalancing decision. Finally, the rebalancing decision is communicated to the routers to program the forwarding plane. In subsequent sections, we describe the requirements with recognition and rebalancing as they pertain to I2RS.

### 1.1. Acronyms

ECMP: Equal Cost Multi-path

GRE: Generic Routing Encapsulation

LAG: Link Aggregation Group

LSR: Label Switch Router

MPLS: Multiprotocol Label Switching

NVGRE: Network Virtualization using Generic Routing Encapsulation

PBR: Policy Based Routing

QoS: Quality of Service

STT: Stateless Transport Tunneling

VXLAN: Virtual Extensible LAN

### 1.2. Terminology

Large flow(s): long-lived large flow(s)

Small flow(s): long-lived small flow(s) and short-lived small/large flow(s)

## 2. Large Flow Recognition and Signaling

### 2.1. Network-based Recognition of Large Flows

The first step is recognizing large flows. There are two ways for recognizing large flows as described in [OPSAWG-large-flow].

The first method is automatic hardware-based recognition in which the large flows are identified in hardware. Once a large flow is recognized, it needs to be communicated to a management entity (such as an SDN controller) that is capable of making rebalancing decisions. This communication is out of scope for I2RS and can be handled using protocols such as IPFIX [RFC 7011].

The next method is where sFlow or IPFIX packet sampling can be used to convey packet samples to an external entity such as sFlow or IPFIX collector. The external entity recognizes large flows and this entity signals the large flows to another management entity that is capable of making rebalancing decisions (such as an SDN controller). Once again, this communication is out of scope of the I2RS.

### 2.2. Application-based Signaling of Large Flows

Instead of having the network recognize large flows, the large flow can be signaled by an application that is known to instantiate large flows, e.g. a backup operation, and may perhaps indicate other parameters such as the latency desired. Such flows would once again need to be signaled to the management entity capable of routing or rebalancing decisions. This communication is also outside the scope of I2RS.

## 3. Flow Rebalancing

### 3.1. Local Rebalancing

In the case of local rebalancing, the utilization of the component links that are part of the LAG or ECMP are monitored and the flows are redistributed among the member links to ensure optimal load balancing across all of the component links. Typically, this involves redirecting large flows to individual ECMP or LAG components, and potentially adjusting the weights used to distribute small flows across these components, using mechanisms specified in [OPSAWG-large-flow].

This approach works regardless of whether the underlying network is IP or MPLS.

To achieve this, there are two requirements for I2RS:

- . For redirecting large flows to a specific member, a PBR entry is required with a key that identifies the flow and a corresponding nexthop that identifies the specific LAG or ECMP component.
- . For adjusting the weights used to distribute traffic across components of the LAG or ECMP, a mechanism is needed that identifies ECMP entries and is able to associate weights that can be programmed for each of the components. To do this in a scalable fashion, it would be useful to have the notion of an ECMP group that is used by multiple routes.

At the RIB level, the nexthop information is typically specified as an outgoing IP interface. However, in an L2/L3 switch, the IP interface may be a VLAN, and in turn there are several "bridge ports" that are members of the VLAN.

Typically, the route entry is resolved to a specific port before the forwarding table is programmed. If the bridge port is a LAG, there will be member ports associated with that LAG. Typically, the resolution down to an individual port is done via means not specified in any standard.

From the standpoint of I2RS, the ability to address individual ports in a router is desirable. This requires the I2RS topology to be aware of LAG members, and the ability of routers to accept route or PBR entries that map to a specific member port within a LAG.

### 3.2. Global Rebalancing

#### 3.2.1. IP Networks

For IP networks, this involves programming a globally optimal path for the large flow. The globally optimal path is programmed in the IP network using hop-by-hop PBR rules.

For IP networks, this involves creating a globally optimal path [HEADERA-dynamic-flow-scheduling] using a network management entity which hosts an I2RS client. The globally optimal path is programmed in the IP network using hop-by-hop PBR rules. The weights of the ECMP table for different nexthops should be adjusted to factor the long-lived large flows - this is explained below with an example.

As an example, consider a 4 way ECMP at node n1 with IP nexthops n11, n12, n13, n14 using links l1, l2, l3, l4 each of capacity 10

Gbps. Say, a long-lived large flow of average bandwidth 2 Gbps is admitted to one of the links l3. The ECMP nexthop table needs to be adjusted to approximately account for the long-lived large flow so that the other flows do not overload link l3 which is already used by the large flow. The ECMP nexthop table will be programmed as  $w1*n11$ ,  $w2*n12$ ,  $w3*n13$ ,  $w4*n14$  where  $w1=w2=w4=1$  and  $w3=0.8$ ; this needs to be done for all the routes using the same set of nexthops.

Now, if there are other set of nexthops from node n1 using link l3, they should also be adjusted. Say, there is another set of IP ECMP nexthops n13, n14, n15, n16 using links l3, l4, l5, l6. The ECMP nexthop table will be programmed as  $w1*n13$ ,  $w2*n14$ ,  $w3*n15$ ,  $w4*n16$  where  $w2=w3=w4=1$  and  $w1=0.8$ ; this needs to be done for all the routes using the same set of nexthops. In practice, there could be multiple large flows on a single link and the ECMP nexthop table must be adjusted to factor all of these flows.

As mentioned in Section 3.1. , it would be useful to have a way of addressing an ECMP group, so that all routes sharing an ECMP group are addressed together.

### 3.2.2. MPLS Networks

There are several ways to address global load rebalancing in MPLS networks. For example:

- . Have multiple LSPs between ingress and egress routers. In this case, having a PBR entry at the edge LSR that forwards the large flow to specific LSP known to have the necessary bandwidth is needed.
- . Program a new LSP for a given large flow.

Here the requirements for I2RS would be providing the ability to program PBR entries at the edge LSR, and the programming new LSPs in the network.

## 4. Operational Considerations

Operational considerations would be similar to those specified in [OPSAWG-large-flow].

## 5. IANA Considerations

None.

## 6. Security Considerations

This draft specifies a use case for I2RS and does not introduce any new security requirements beyond those already under consideration for I2RS.

## 7. Acknowledgements

The authors would like to sincerely acknowledge Dan Romascanu for his help with understanding the relationship between the Interfaces MIB and LAG. The authors would also like to thank Prasad Jogalekar, Mukhtiar Shaikh and Muhammad Durrani for the discussions on the topic of global load balancing.

## 8. References

### 8.1. Normative References

### 8.2. Informative References

[OPSAWG-large-flow] Krishnan, R. et al., "Mechanisms for Optimal LAG/ECMP Component Link Utilization in Networks," February 2014.

[HEDERA-dynamic-flow-scheduling] Al-Fares, M. et al., "Hedera: Dynamic Flow Scheduling for Data Center Networks", December 2009

[sFlow-v5] Phaal, P. and M. Lavine, "sFlow version 5," July 2004.

[RFC 7011] Claise, B., "Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information," September 2013

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

## Authors' Addresses

Ram Krishnan  
Brocade Communications  
ramk@brocade.com

Anoop Ghanwani  
Dell  
anoop@alumni.duke.edu

Sriganesh Kini  
Ericsson



Internet-Draft

I2RS Large Flow

September 2013

sriganesh.kini@ericsson.com

Dave Mcdysan

Verizon

dave.mcdysan@verizon.com



Network Working Group  
Internet-Draft  
Intended status: Standards Track  
Expires: April 23, 2014

R. Fernando  
J. Medved  
Cisco Systems  
E. Crabbe  
Google  
K. Patel  
Cisco Systems  
October 20, 2013

I2RS Protocol Requirements  
draft-rfernando-i2rs-protocol-requirements-00

Abstract

The Interface to Routing System (I2RS) allows an application to programmatically query and modify the state of the network. This document defines requirements for an I2RS protocol between applications (clients) and network elements (servers).

Requirements Language

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].

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 April 23, 2014.

## Copyright Notice

Copyright (c) 2013 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

1. Introduction . . . . .	2
2. Terminology . . . . .	3
3. I2RS Protocol High Level Design Objectives . . . . .	5
4. I2RS Protocol Requirements . . . . .	7
4.1. General Assumptions . . . . .	7
4.2. Transport Requirements . . . . .	8
4.3. Identity Requirements . . . . .	10
4.4. Message Encoding Requirements . . . . .	11
4.5. Message Exchange Pattern Requirements . . . . .	11
4.6. API Method Requirements . . . . .	13
4.7. Service and SDM Requirements . . . . .	14
4.8. Security Requirements . . . . .	16
4.9. Performance and Scale Requirements . . . . .	17
4.10. High Availability Requirements . . . . .	18
4.11. Application Programmability Requirements . . . . .	19
4.12. Operational Requirements . . . . .	19
5. Contributing Authors . . . . .	20
6. References . . . . .	20
6.1. Normative References . . . . .	20
6.2. Informative References . . . . .	20
Authors' Addresses . . . . .	21

## 1. Introduction

I2RS defines a standard, programmatic interface for state transfer in and out of the Internet's routing system. I2RS is intended to automate network operations - it will allow applications to quickly interact with routing systems and to implement more complex operations, such as policy-based controls. [I-D.ietf-i2rs-problem-statement] gives detailed problem statement for I2RS, while [I-D.ietf-i2rs-architecture] defines its high-level architecture.

This document expands on the required aspects of a protocol for I2RS, described in Section 5 of [I-D.ietf-i2rs-problem-statement], and refines I2RS protocol requirements formulated in Section 6 of [I-D.ietf-i2rs-architecture]. The purpose of this document is as follows:

1. To help stakeholders (equipment vendors, operators, application programmers or interested IETF participants), to arrive at a common understanding of the critical characteristics of the I2RS protocol(s).
2. To provide requirements to designers of the I2RS framework and protocols on aspects of the framework that warrant significant consideration during the design process.
3. To allow the stakeholders to evaluate technology choices that are suitable for I2RS, to identify gaps in these technologies and to help evolve them to suite I2RS's needs.

## 2. Terminology

This document uses the following terminology definitions.

**Service:** For the purposes of I2RS, a service refers to a set of related state access functions together with the policies that control it's usage. For example, the 'RIB service' is a service that provides access to state held in a device's RIB.

**Server:** Is a system that implements one or more services that can be accessed by client systems via well defined interfaces (APIs). A server can export multiple services. A server is typically a network device. [ed: as an aside, I still find this usage to be completely counterintuitive]

**Client:** A system that uses a service implemented by a server through a well defined interface. A client can make use of multiple services from multiple servers.

**Participants:** The server and client are collectively referred to as the participants of a service.

**Transport:** Any end-to-end mode of communication between a server and a client that allows exchange data the exchange of data. In principle, the transport hides the topology and other network properties from the participants of a service.

**Messages:** Messages are logical units of data that are exchanged between service participants.

**Message Exchange Pattern:** A categorization of a way in which messages could be exchanged between service participants. MEPS specify the sequence, order, direction and cardinality of messages exchanged. Request-response and asynchronous notifications are examples of MEPS.

**Message Data Model:** The schema representing the structure of messages being exchanged between the service participants. The MDMs can specify certain constraints such as data type, length, format and valid values of fields in messages.

**Message Encoding:** The "wire" representation of messages exchanged between service participants.

**API Method:** Is an application level procedure or a function that is invoked by the client to query or modify the state held in the server.

**Service Scope:** Is the functional scope of a service. The service scope is established during the service definition phase.  
[definition needs work]

**Service Data Model:** The schema representing the conceptual structure of the state held in the server for a given service. The SDMs can specify certain constraints such as the data type, length, format and allowed values for fields representing the state. They also describe the relationship between various state elements that constitute the state.

**Modeling Language:** A language that defines schema for Message Data Models and Service Data Models.

**Namespaces:** A method for uniquely identifying and scoping of schemas declared for messages and services. Namespace is an important consideration when defining services and messages.

**Service State or State:** Is the general data held by the server for a given service.

**State Element:** A readable or writable state present in the server. The term 'State Element' may refer to states at different levels of granularity.

**State Identifier:** A unique identity for the state element. The identifier is derived from the SDM and uses the same naming convention as the SDM. State Identifiers may be viewed as the 'key' for the state.

**State Value or 'value':** A value that is assigned to a particular state identifier (key). The state is referred to using the State Identifier or 'key' in operations that sets or transfers the value of the state.

**State Owner:** Identity of the client that was the source of a state held in the server.

**State lifetime:** The duration which the state is maintained in the server.

**Datastore:** The physical mechanism used to store a service's state.

**Capabilities:** Capabilities represents the functionality supported by a server including the services supported and exported to clients.

**Authentication:** A mechanism that allows a server to recognize the identity of a client.

**Authorization:** A mechanism that allows determination of what an authenticated client is allowed to do.

**Confidentiality:** Specifies if data remains confidential while in transit between service participants.

**Policy:** For the purposes of this document, a policy is an explicit user configurable modification to the default behavior of the system. The enforcement could be conditional; a given policy could potentially become active only when certain conditions are met.

### 3. I2RS Protocol High Level Design Objectives

The core guiding principles and objectives that should be used in defining the specifics of the I2RS protocol(s) are as follows:

- HLO-1. Design for Scale and Performance: is a key criteria for making design choices. The design should meet current and future performance and scale needs. Design patterns that allow us to compose a scalable, high performing system are well understood and should be utilized where possible.
- HLO-2. Extensible: I2RS will be deployed in environments that will evolve over time. Hence the system should be designed with provisions that will allow significant modifications/extensions to be added to existing mechanisms. An extensible and future-proof design will drive better adoption as it is a promise against future technology churn.
- HLO-3. Promote Reuse: Reuse in this context refers to using existing tools, technologies and mechanisms instead of inventing them from scratch. It also refers to reusing a network device's current set of capabilities that applications could harness without reinventing them from scratch.
- HLO-4. Promote Portability: Portability refers to the ease with which software written for one device or environment can be moved to work seamlessly with another device or environment to achieve similar functionality. A fundamental requirement for I2RS is to achieve predictive and consistent behavior when applications are migrated from one platform or environment to another.
- HLO-5. Security: I2RS may be deployed in environments where it might be subjected to threats and denial-of-service attacks that might cause intentional damage to the functioning of a network. This could be in the form of loss of service, degradation of performance, loss of confidentiality, etc. Therefore, the I2RS protocol must provide basic security mechanisms including but not limited to authentication, authorization, confidentiality along with sufficiently expressive policy requirements for each.
- HLO-6. Separation of concerns: The components of the system should be decoupled from each other as much as possible to achieve clear separation of concerns. This modularity would allow for interchangeable design and implementation choices that address the individual components requirements.
- HLO-7. Robustness: Robustness is the ability of a system to operate in the face of failures and errors. It is also its ability to correctly and predictably recover from such errors and to settle to a known state. Since applications that use the I2RS framework are remote and would be controlling the entire network, ensuring fault tolerance is an important consideration.



Most of these requirements cut across all the components of the system and hence should be kept in mind while designing each component and the system as a whole.

#### 4. I2RS Protocol Requirements

This section is divided into multiple sub-sections, each dealing with a specific consideration of I2RS protocol design. As we list the requirements under each subsection, we'll annotate each requirement with what high level objectives they meet. Additional reasoning is additionally provided where appropriate.

##### 4.1. General Assumptions

This section captures the general, high level assumptions of the I2RS framework. The context for defining protocol requirements is shown in the following figure.

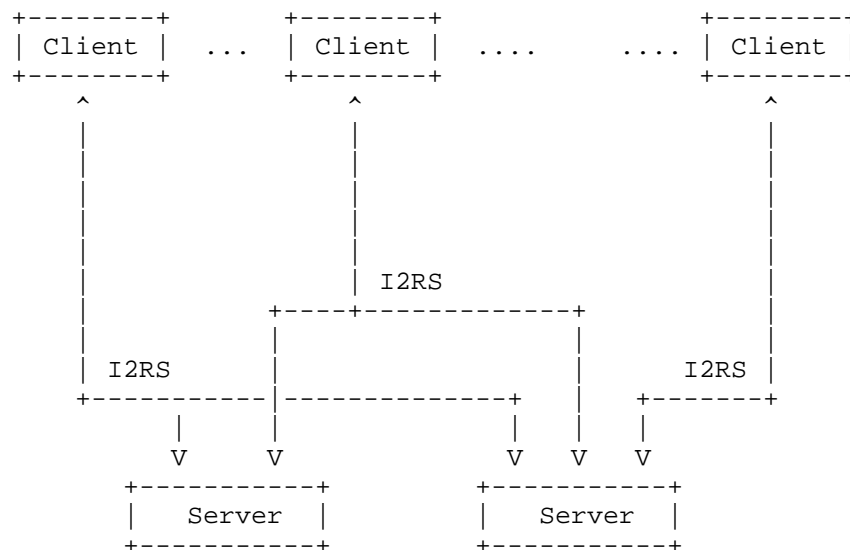


Figure 1: Protocol requirements context

Since the design choices for the I2RS framework are many, some simplifying assumptions could make the framework requirements more tangible and useful.

GEN-1: Programmatic access to the state held in a network device is provided to an application by exposing a set of API's from the device to the application. Due to this characteristic, I2RS is a client-server protocol/framework. I2RS must provide mechanisms for the client to discover services that a server provides.

GEN-2: The client can use the APIs provided by the server to programmatically add, modify, delete and query state held in the server. Additionally clients can register for certain events and be notified when those events occur.

GEN-3: The client and the server communicate using a simple transport connection. The client initiates the transport connection to the server. The server does not know the number and timing of the connections from its clients.

GEN-4: A service provides access to the state held in the server structured according to the SDM of that service. A service allows a client the ability to manipulate the service state.

GEN-5: The I2RS MUST define a data model to describe the SDMs supported in the server and MUST define a data modeling language to formally describe that data model. I2RS MUST specify the mapping from the service data model to the message data model and subsequently to the client API.

#### 4.2. Transport Requirements

The transport layer provides connectivity between the client and the server. This section details the transport requirements.

TR-1: There should exist a default transport connection between the client and the server for communication. This control connection is point-to-point and should provide in-order and reliable delivery of data in both directions. The simplest I2RS setup would only have a single transport session between the participants.

TR-2: Depending on the data being exchanged, there may be additional transport connections between the client and server defined in future. The characteristics of these additional transport connections will be dictated by the requirements that create them.

TR-3: The transport connection between the client and server MUST have mechanisms to support authentication, authorization and to optionally provide confidentiality of data exchanged between the client and the server. See Section 4.8 for more details.

- TR-4: A client could connect to multiple servers. Similarly, a server could accept connections from multiple clients. Any participant may initiate the transport connection.
- TR-5: The exact technology used for the transport layer should be replaceable. There should be a single mandatory transport that must be supported by all participants. This requirement will ensure that there is always an interoperable transport mechanism between any client and any server.
- TR-6: Clients and servers by default communicate using a point-to-point transport connection.
- TR-7: Point-to-multipoint transports are mainly used to scale the system by avoiding ingress replication when the same message has to be delivered to multiple receivers. P2MP transport would work hand-in-hand with a P2MP MEP. The subject of P2MP transport and P2MP MEP is for future work.
- TR-8: Once the transport connection is established, it is desirable to reuse it to perform multiple operations. This requirement ensures that the system scales by amortizing the session setup cost across multiple operations. Session down events may not have an impact on the state maintained by the server.
- TR-9: After the transport is established, the participants exchange capabilities and other session parameters before exchanging service related messages.
- TR-10: Messages pertaining to multiple services could be exchanged over a single transport connection.
- TR-11: The "default" transport connection between a client and a server is purely for control plane message exchanges. Data plane packets are not expected to be sent over this "default" connection. When required, packets to be extracted from or injected to the data plane could be designed as a service in itself that sets up a separate packet injection/extraction channel that provides the correct characteristics.
- TR-12: For operational reasons, a participant MUST be able to detect a transport connection failure. To satisfy this requirement, transport level keep-alives could be used. If the underlying transport connection does not provide a keep-alive mechanism, it should be provided at the I2RS protocol level. For example, if TCP is used as a transport, TCP keep-alives could be used to detect transport session failures.

#### 4.3. Identity Requirements

I2RS could be used in a multi-domain distributed environment. Therefore a fool-proof way to ascertain the identity of clients is of utmost importance. Identity provides authenticated access to clients to state held by the server.

ID-1: Each client should have a unique identity that can be verified by the server. The authentication could be direct or through an identity broker.

ID-2: The server should use the client's identity to track state provided by the client. State ownership enables multiple clients to edit their shared state. This is useful for troubleshooting and during client death or disconnection when the client's state may have to be purged or delegated to another client that shares the same identity.

ID-3: The client's I2RS identity should be independent of the location or the network address of the physical node in which it is hosted. This allows the client to move between physical nodes. It also allows a standby client to take over when the primary fails and allows shared state editing by multiple clients as discussed in I.2.

ID-4: A client that reboots or reconnects after a disconnection MUST have the same I2RS identity if it wishes to continue to operate on the state that it previously injected.

ID-5: A clients ability to operate on a state held by the server is expressed at the granularity of a service. A service could be read-only or read-write by a client possessing a particular identity.

ID-6: A policy on the server could dictate the services that could be exposed to clients. Upon identity verification, the authorized services are exported to the client by capability announcement.

ID-7: A client can edit (write, delete) only the state that was injected by it or other clients with the same shared identity. Therefore, two conditions must be met for a client to edit a state through a session. First, the client should receive capability from the server that it has 'edit' permissions for the service in question, and, secondly, the state that it edits should be its own state.

ID-8: The server retains the client's identity till all of that client's state is purged from the server.

#### 4.4. Message Encoding Requirements

Clients and servers communicate by exchanging messages between them. Message encoding is the process of converting information content in a message to a form that can be transferred between them.

ME-1: Every message between the client and the server is encoded in a transport-independent frame format.

ME-2: Each message is serialized on the sender's side and de-serialized on the receiver's side. The technology used for encoding and decoding messages could be negotiated between the client and the server.

ME-3: A mandatory default encoding standard should be specified and implemented by all I2RS participants. This ensures that there is an interoperable default encoding mechanism between any client and any server.

ME-4: The mandatory encoding technology chosen should be well supported by a developer community and should be standards based. Availability of tools and language bindings should be one of the criteria used in selecting the mandatory encoding technology.

ME-5: If multiple message encoding is supported in the framework, the encoding used for the current session should be configured using a policy on the server side and negotiated using capabilities. Note that currently there is no requirement to support multiple encoding schemes.

ME-6: The message encoding standard should be language and platform neutral. It should provide tools to express individual fields in a message in a platform independent IDL-based language.

ME-7: The encoding/decoding mechanism should be fast and efficient. It should allow for operation on legacy equipment.

ME-8: The encoding scheme should allow for optional fields and backward compatibility. It should be independent of the transport and the message exchange pattern used.

ME-9: Human readability of messages exchanged on the wire might be a goal but it is secondary to efficiency concerns.

#### 4.5. Message Exchange Pattern Requirements

Message exchange patterns form the basis for all service level activities. MEPs create a pattern of message exchanges that any task

can be mapped to whether initiated by a client or the server. This section provides the requirements for MEPS.

MEP-1: I2RS defines three types of messages between the client and the server. First, capabilities need to be exchanged on session establishment. Second, API commands are sent from a client to a server to add, delete, modify and query state. And third, asynchronous notifications are sent from a server to a client when interesting state changes occur.

MEP-2: The above message exchanges can be satisfied by two message exchange patterns. Capabilities and asynchronous notifications can be satisfied by one-way unsolicited fire and forget message. API commands can be satisfied using a request-response message exchange. The base I2RS framework should thus support at least these two MEPS.

MEP-3: For a request-response MEP, the server SHOULD acknowledge every request message from the client with a response message.

MEP-4: The response message in a request-response MEP SHOULD indicate that the server has received the message, done some basic sanity checking on its contents and has accepted the message. The arrival of a response does not mean all post processing of the message has completed.

MEP-5: If an error occurs with an API command, The response message MUST indicate an error and carry error information if there was a failure to process the request. The error code SHOULD be accompanied by a descriptive reason for the failure.

MEP-6: Error codes SHOULD indicate to the client which layer generated that error (transport, message parsing, schema validation, application level failure, etc). The I2RS framework should specify a standard set of error codes.

MEP-7: The request-response messages SHOULD be asynchronous. That is, the client should not be required to stop-and-wait for one message to be acknowledged before it transmits the next request. [ed: there must be a method for dependency tracking in the protocol. possibly negotiate as an optional capability?]

MEP-8: To satisfy asynchronous operations, a mechanism MUST exist to correlate response messages to their respective original requests. For example, a message-id could be carried in the response that would help the sender to correlate the response message to its original request.

MEP-9: The response messages need not arrive in the order in which the request was transmitted. [ed: again, there must be a method for providing an order of operations, and for receiving positive ack on completion]

MEP-10: The request message SHOULD carry an application cookie that should be returned back to it in the corresponding response. [ed: provide further justification]

MEP-11: Besides the request-response MEP, there is a need for a fire and forget MEP. Asynchronous notifications from the server to the client could be carried using this MEP. Fire and forget MEPs can be used in both client-to-server and server-to-client directions.

MEP-12: Fire-and-forget messages MAY optionally be acknowledged.

MEP-13: The fire-and-forget MEP does not carry a message-id but it SHOULD carry a cookie that can be set by the sender and processed by the receiver. The cookie could help the receiver of the message to use the message for its intended purpose.

#### 4.6. API Method Requirements

API methods specify the exact operation that one participant intends to perform. This section outlines the requirements for API methods.

API-1: The I2RS framework SHOULD provide for a simple set of API methods, invoked from the client to the server. These methods should allow to add, modify, query and delete of state that the server maintains.

API-2: The I2RS framework should provide for three query methods, subscribe, unsubscribe, and one-time-query, that the client can use to express an interest or collect specific state changes or state from the server.

API-3: The API methods discussed in API-1 and API-2 should be transported in a request-response MEP from the client to the server.

API-4: The API framework SHOULD provide for a single notify method from the server to the client when interested state changes occur. The notification method SHOULD be transported in a fire-and-forget MEP from the server to the client.

API-5: The framework SHOULD define a set of base API methods for manipulating state. These SHOULD be generic and SHOULD not be service specific.

API-6: All API methods that affect the state in the server SHOULD be idempotent. That is, the final state on the server should be independent of the number of times a state change method with the same parameters was invoked by the client.

API-7: All API methods SHOULD support a batched mode for efficiency purposes. In this mode multiple state entries are transmitted in a single message with a single operation such as add, delete, etc. For methods described in A.1 and A.2 which elicit a response, the failure mechanism that is specific to a subset of state in the batch should be devised. The Notify method SHOULD also support a batched mode.

API-8: Since the API methods are primarily oriented towards state transfer between the client and server, there SHOULD be a identifier (or a key) to uniquely identify the state being addressed.

API-9: API methods that refer to value of a particular state SHOULD carry the state identifier (key) as well as the its value. For instance, during a state add operation, both the identifier (key) and the value SHOULD be passed down from the client to the server.

API-10: Besides basic API methods common to all services, a server could support proprietary methods or service specific methods. The framework SHOULD specify a mechanism to express these methods and their semantics through a modeling language or otherwise. The ability to support additional API methods SHOULD be conveyed to the client through capability negotiation.

API-11: Transactions allow a set of operations to be completed atomically (all or nothing) and that the end result is consistent. Transactions MAY be required for some network applications. [ed: i'd like to remove this for the the first version. it's a bit of a canard']

#### 4.7. Service and SDM Requirements

SVC-1: Each service is associated with a service data model that defines the type and structure of the state (data) pertaining to that service. I2RS MUST provide mechanisms to manage the state held in the server in accordance to the SDM.

SVC-2: The base I2RS API methods MUST allow a client to add, modify, query and delete state information.

SVC-3: Neither the transport or the MEP SHOULD have any bearing on the structure of the state being transferred. Each service module



in the server would be responsible for interpreting the structure of the state being transferred corresponding to the SDM.

SVC-4: A client, after proper identification, could operate on multiple 'services' that are exported to it. A client could have read-only or read-write access to a given service. The availability of the service, valid entities and parameter ranges associated with same are expressed via the exchange of capability information with the client.

SVC-5: The arrangement and structure of state (SDM) SHOULD be expressed in a network friendly data modeling language. [ed: I have no idea what this means]

SVC-6: The data modeling language SHOULD have the ability to express one-to-one, one-to-many and hierarchical relationships between modelled entities.

SVC-7: The data modeling language MUST allow for a service data model to be extended beyond its initial definition. The language MUST be able to express mandatory and optional elements in SDMs. It SHOULD also have the ability to express exceptions for unsupported elements in the model.

SVC-8: For every service that it wishes to expose to a client, the server SHOULD send capabilities that indicate the existence and identity of the service data model, as well as valid states and parameter ranges, any exceptions to it and the optional features of the data model that it supports.

SVC-9: The data modeling language MUST be able to express a dependence of service data model on another SDM. It SHOULD also have the ability to express references to state elements in another service data model.

SVC-10: The modeling language MUST have the ability to express read/write and read-only constraints for state elements in an SDM. Readable state elements are populated and managed by the server and clients don't have the ability to write their value. Routing next-hops added by a client is an example of read-write state. Statistics associated with that next-hop is an example of read-only state.

SVC-11: Query and notification APIs MUST be able to carry both read-only as well as read-write state.

SVC-12: Besides specifying a SDM, a service SHOULD also specify the interesting state changes that clients can subscribe to for

notifications. The absence of such a specification SHOULD be interpreted as an implicit unavailability of any subscribable entities. [ed: terminology to be worked out here]

SVC-13: A client which is authenticated to access a service (either read-only or read-write) MAY subscribe to any subscribable state change events.

SVC-14: A subscribe method SHOULD optionally have a filter associated. This increases the efficiency by filtering out events that the client is not interested in. The notification filter should have the ability to express state identifiers and wildcards for values.

SVC-15: The base API operations MUST be generic and allow a client to operate on multiple services with the same set of methods. Each service dictates its one schema or SDM.

SVC-16: I2RS protocol SHOULD allow a server to export standard services as well as vendor proprietary services. A namespace scheme should be devised to recognize standard and proprietary services.

SVC-17: The server SHOULD indicate to the client the availability of infrastructure to manage the state that it maintains. This includes but is not limited to the availability of persistent store, the availability of timer to clean up state after a specified timeout, the ability to clean up state on the occurrence of an event, etc. Equipped with this information, the client is responsible for the lifetime of the state.

SVC-18: Each state SHOULD have a set of meta data associated with it. This includes the state's owner, the state's lifetime attributes, a creation and modification timestamp, etc. This information would aid in the debugging of the system. An authenticated client that is exposed to a service SHOULD also have access to the meta data associated with that service's state.

#### 4.8. Security Requirements

This section lists security requirements for the I2RS protocol.

SEC-1: Every client MUST be authenticated and associated with an identity. A secure mechanism to uniquely identify a client such as certificates MUST be adopted.

SEC-2: Every client MUST have an authorized role whereby only certain state can be accessed and only certain operations can be

performed by that client. To keep the model simple and applications portable, authorization MUST be at a per service level and not on individual state element level.

SEC-3: The framework MUST provide for information confidentiality and information integrity as options.

SEC-4: Every state maintained by the server SHOULD be tagged with the client's identity as well as meta-data to indicate last access and last modifications time-stamps. This ensures accountability and helps auditing the system.

SEC-5: Mechanisms to "hook" into third-party security infrastructure whenever possible SHOULD be provided in order to achieve security goals. Applications programmers SHOULD be kept free of security concerns and yet given a flexible, configurable and well integrated security model.

#### 4.9. Performance and Scale Requirements

Performance requirements are usually woven in with the functional requirements of a system. They feature in every decision made to fulfill the systems requirements. Performance and scale are a complex function of many things. Hence performance requirements cannot be precisely quantified by a single number. This section lays out some common sense guidelines that should be kept in mind while designing the system from a scale and performance standpoint.

PS-1: The request-response MEP SHOULD be asynchronous. This ensures that a system is not stuck waiting for a response and makes the entire system more responsive and enables concurrency between operations.

PS-2: When applicable, messages SHOULD carry application level cookies that enable an application to quickly lookup the context necessary to process a message. Management of cookies is the applications responsibility.

PS-3: The framework SHOULD allow for bulk operations which amortizes the communication and messaging costs.

PS-4: The transport protocol SHOULD provide for a binary encoding option for messages between the participants.

PS-5: The transport protocol MUST provide for both encrypted and non-encrypted transport between participants.

- PS-6: The transport protocol MUST provide for message prioritization.
- PS-7: Multiple operations should be completed using a single transport session whenever possible.
- PS-8: The server MUST be kept stateless with respect to the number and location of each client.
- PS-9: Filtered subscription MUST be supported for notifications.
- PS-10: An efficient synchronization mechanism between a client and a server SHOULD be provided that avoids transferring all the state between them.
- PS-11: Allow clients that perform infrequent operations to disconnect their transport connection without cleaning up their state.
- PS-12: Create the basic necessary mechanisms in the framework and build everything else as a service if possible.

#### 4.10. High Availability Requirements

The ability of the system to withstand operational failures and function in a predictable manner is called availability. A few guidelines that are important are,

- HA-1: A 'superuser' identity SHOULD be provided that is capable of changing security policies, clearing state and perform actions that override client initiated actions in the system.
- HA-2: Session disconnection and client deaths MUST be handled gracefully. They should have the least impact on the server system.
- HA-3: Client connections and disconnections MUST be logged and provided as a well-known service to authenticated users.
- HA-4: Clients MUST be notified of message processing and other errors through error codes in messages.
- HA-5: A mechanism to gracefully terminate the session between the client and the server MUST be provided.
- HA-6: A mechanism for authenticated clients to query the load attributes of the system, both instantaneous and running average MUST be provided as a service.

#### 4.11. Application Programmability Requirements

The framework should pay particular attention the requirements of application programmers. A well written framework should improve the productivity of programmers and shorten the time to make an application. This section has some issues to consider when devising the framework from an applications standpoint.

PGM-1: A client programming framework SHOULD allow applications writers to focus on the app functionality rather than mechanisms required to communicate with the server.

PGM-2: The application once written to certain requirements should be portable to other identical environments. The framework should not have fine grained data access controls as this would lead to a poorly written application with portability issues.

PGM-3: The framework should be devised in a manner that it is possible to automate code generation and constraint checking in popular programming languages. Generated code can then be used readily by application programmers instead of dealing with the nitty-gritties of the system.

PGM-4: Define a common repository for SDMs from which clients can obtain the SDMs they are interested in and automatically generate most of the boilerplate code.

PGM-5: Provisions SHOULD be made for debugging and troubleshooting tools that includes message trace, call traces, access to relevant server traces and logs, packet decode tools to trace and decode messages on the wire, consistency checkers of state inserted into a server.

PGM-6: The toolset should have a general portion (for common functions, such as session management) and SDM-specific portions (for example, a flag to control generation of debug code in code generated for a particular SDM).

PGM-7: The framework SHOULD define SDMs and MDMs in a language neutral format so as to enable code generation in multiple programming languages.

#### 4.12. Operational Requirements

OP-1: There is a need to identify operational performance parameters of the system and provide mechanisms to retrieve them from a running system.

OP-2: Provide a way to upgrade a service independently of the other services. This modularity allows uninterrupted operation of the all but one service which is being upgraded.

OP-3: Provide a detailed workflow for bringing about a new service. This workflow will start with the need to introduce a new service and address the following: How SDMs defined? Where are they standardized? How are new entities (MEPs, transport, encoding) introduced? What are the tools and workflow involved to develop and operationalize a service. The intent is to introduce a level of understanding about stakeholders responsibilities.

OP-4: Provide mechanisms and methodologies to test a new service before deployment.

## 5. Contributing Authors

Thanks to the following people for reviewing and providing meaningful contributions: Alia Atlas, Palani Chinnakannan, Alexander Clemm, Muthumayan Mahdhayyan, John McDowell and Bruno Rijsman and David Ward.

## 6. References

### 6.1. Normative References

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

### 6.2. Informative References

[I-D.ietf-i2rs-architecture]  
Atlas, A., Halpern, J., Hares, S., Ward, D., and T. Nadeau, "An Architecture for the Interface to the Routing System", draft-ietf-i2rs-architecture-00 (work in progress), August 2013.

[I-D.ietf-i2rs-problem-statement]  
Atlas, A., Nadeau, T., and D. Ward, "Interface to the Routing System Problem Statement", draft-ietf-i2rs-problem-statement-00 (work in progress), August 2013.

[RFC1776] Crocker, S., "The Address is the Message", RFC 1776, April 1995.

[RFC1925] Callon, R., "The Twelve Networking Truths", RFC 1925, April 1996.

Authors' Addresses

Rex Fernando  
Cisco Systems  
170 W Tasman Dr,  
San Jose, CA 95134  
US

Email: [rex@cisco.com](mailto:rex@cisco.com)

Jan Medved  
Cisco Systems  
170 W Tasman Dr,  
San Jose, CA 95134  
US

Email: [jmedved@cisco.com](mailto:jmedved@cisco.com)

Edward Crabbe  
Google  
1600 Amphitheater Parkway  
Mountain View, CA 94043  
US

Email: [edward.crabbe@gmail.com](mailto:edward.crabbe@gmail.com)

Keyur Patel  
Cisco Systems  
170 W Tasman Dr,  
San Jose, CA 95134  
US

Email: [keyur@cisco.com](mailto:keyur@cisco.com)

Network Working Group  
Internet-Draft  
Intended status: Informational  
Expires: April 24, 2014

S. Whyte  
Google Inc.  
M. Hines  
W. Kumari  
Google, Inc.  
October 21, 2013

Bulk Network Data Collection System  
draft-swhyte-i2rs-data-collection-system-00

Abstract

Collecting large amounts of data from network infrastructure devices has never been very easy. Existing methods generate CPU and memory loads that may be unacceptable, the output varies across implementations and can be difficult to parse, and these methods are often difficult to scale. I2RS programmatic interfacing with the routing system may exacerbate this problem: state needs to be collected from nodes and fed to consumers participating in the control plane that may not be physically close to the nodes. This state includes not only control plane information, but elements of the data plane that have a direct impact on control plane behavior, like traffic engineering.

This document outlines a set of use cases requiring a flexible framework to collect routing system data, and the features and functionality needed to make such a framework useful for these use cases.

Requirements Language

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 RFC 2119 [RFC2119].

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 April 24, 2014.

#### Copyright Notice

Copyright (c) 2013 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

#### Table of Contents

1. Introduction . . . . .	3
2. Desired functionality . . . . .	3
2.1. Database Model . . . . .	4
2.2. Pub-Sub . . . . .	4
2.3. Capability Negotiation . . . . .	5
2.4. Format Agnostic . . . . .	5
2.5. Transport Options . . . . .	5
2.6. Filtering . . . . .	5
2.7. Timestamps . . . . .	6
2.8. Introspection . . . . .	6
2.9. Registration . . . . .	6
3. Use cases . . . . .	6
3.1. Push . . . . .	7
3.1.1. Interface counters . . . . .	7
3.1.2. Thresholds . . . . .	7
3.1.3. Streaming . . . . .	7
3.2. Pull . . . . .	7
3.2.1. Interface counters . . . . .	8
3.2.2. RIB Dump . . . . .	8
3.2.3. Arbitrary data collection . . . . .	8
3.3. Dynamic subscriptions . . . . .	8
4. Subscriber versus consumer . . . . .	8
4.1. Remapping . . . . .	8
5. Errors . . . . .	9
6. IANA Considerations . . . . .	9
7. Security Considerations . . . . .	9

8. Acknowledgements	9
9. References	10
9.1. Normative References	10
9.2. Informative References	10
Authors' Addresses	10

## 1. Introduction

Managing and monitoring a network requires getting state out of it. You can't manage what you don't measure, as the saying goes. Currently there are a limited set of tools to get data off of network nodes, and they do not lend themselves to programmatic access.

The primary tool today is SNMP. SNMP can be used to both push data off a node (via traps/notifications) and pull data off the box (via queries). SNMP queries have a variety of issues, not the least of which is the fact that the protocol specification requires data structures to be created on demand on network nodes that do not match how the device's operating system data structures store the same data. Fixing this problem has the immediate benefit of reducing CPU and memory consumption of the monitored network devices, greatly increasing the deployability and relevance of a solution. SNMP traps /notifications suffer from a lack of introspection; the network management system (NMS) must be preconfigured to understand what information is being reported.

Other tools include CLI scraping and Syslog. CLI scraping is a low-level pull mechanism and essentially the opposite of programmatic access. Any change in CLI implementation, whether its a simple whitespace correction, re-ordering of configuration stanzas, typographical errors, or even unit changes, can require a rewriting of monitoring software. This is compounded by the fact there is no standardized CLI specification, such that a network with multiple vendors in it requires these rewrites per vendor CLI change.

Syslog is another way to push data off of a network node. Syslog has been around a long time, and while current standards provide structured data output, very few implementations exist on network nodes currently. For the most part NMSes must be trained how to consume and interpret different implementations of syslog.

## 2. Desired functionality

Collecting large data sets with high frequency and resolution, with minimal impact to a device's CPU and memory, is the primary objective. Aspects of the over-all data collection system, such as availability or reliability or scaling, are outside of scope as they deal with the data once it has left the network node.

We are only focusing on getting data off the node in an easily machine parsable format.

### 2.1. Database Model

A database model is desired, whereby a network node can describe the data it has available, and the structure of that data. This gives the implementor the ability to present a database model that can be optimal with the node's internal data structure implementations. The NMS consumes and understands the database model only after it has been trained to do so by incorporating a published version of the database model from the vendor.

It should be noted that all existing data collection methods outlined earlier require explicit knowledge of the method's implementation for integration into a NMS. We do not propose a solution that eliminates this, because heterogeneity of the data is not required, as we can see from existing implementations. Rather, capability negotiation and flexible formats and transports, outlined below, are desired enabling the primary objective of getting large data sets off the nodes with as little impact as possible.

### 2.2. Pub-Sub

An underlying pub-sub model is desired for a variety of features. It provides a security model for authorization, it supports intermediaries allowing the system to scale as needed, and it provides both push and pull methods of data distribution.

In the context of this draft, a pub-sub model is a general concept indicating information flow. Specific system details are obviously critical yet belong in a data model document. The high level desire is to have network nodes as publishers, with an NMS implementing subscribers. Conceptually, they are connected by a message bus, a layer of indirection between the publishers and subscribers. Having a message bus allows publisher fan-in, subscriber fan-out, and a number of other useful features outside the scope of this document. The message bus is frequently referred to as a broker inside pub-sub models.

Having a message bus abstraction allows for considerable flexibility in NMS design as well. Placement of brokers in the network, their redundancy, availability, scaling per publisher or subscriber, can all be tailored to suit an individual network's needs, from extremely simple (flat) to extremely complex with multiple layers of hierarchy. Many implementations of pub-sub models exist, scaling both in number of subscriptions and in number of messages, both of which should be considered carefully in the I2RS context.

### 2.3. Capability Negotiation

Capability negotiation allows a node to inform a subscriber of a number of options. Two extremely important options would be transport protocols and formats supported. Other aspects such as security options and error handling would also be negotiated during this phase.

The capability negotiation phase is done via a control channel opened for the purpose of registering subscriptions with the node. This control channel should be TCP.

### 2.4. Format Agnostic

From the I2RS perspective, this framework should be format agnostic. If a node advertises the ability to present data in XML and the subscriber agrees, then XML can be used. Other formats that have interest are JSON, HTML, and protobufs. Even interest for /proc/net formatted output exists, and would help a NMS based on this framework integrate into existing server configuration management systems.

[ Editor note: even ASN.1 should be an acceptable format. This would potentially allow an extremely easy deployment into an existing SNMP based NMS.]

### 2.5. Transport Options

Because the focus of this framework ends at getting data off the box as quickly as possible, implementations should have the freedom to choose a transport that meets their system design needs and not be restricted by a specific format.

During the negotiation phase a node should advertise all the transport options it provides and allow the subscriber to select what it needs.

Given the time-value of different data elements coming off the node can be quite different, it should be possible to request multiple transports and associate a subscription with the transport protocol of choice.

### 2.6. Filtering

Once a network node has provided its database model to a subscriber, the subscriber needs a way to select parts of the model for subscription, and it needs to be able to request multiple subscriptions at a time.

This framework should provide a standard filtering mechanism so that, independent of the database model structure and contents, a subscriber can select interesting items to collect and bucket them based on standard parameters such as frequency of collection, underlying transport required, whether the data is to be pushed or pulled, or even streaming or one-shot.

### 2.7. Timestamps

Every piece of data collected by this framework needs a timestamp associated with it indicating when the node made it available for collection. This is not required on a per-variable basis, for example data organized into a table only requires a timestamp associated with the table.

This is not to say additional timestamps are not useful for certain data sets nor that other timestamps with other semantics, for example collection time versus advertisement time, can not be used, but rather those additional timestamps are better placed in the database model supported by the device.

### 2.8. Introspection

This framework should support introspection of the database model. Introspection provides support for data verification, easier inclusion of legacy data, and easier merging of data stream.

### 2.9. Registration

After capabilities and a database model have been exchanged, and a filter used to select elements of the model to subscribe to, the framework should support a standard way to register for all the data desired, using whatever capabilities were advertised by the node.

Once registration is complete, the control channel can be closed. Ensuring subscriptions are correct, complete, and replicated or not, is up to the overall system and not the network node.

## 3. Use cases

Following are example use cases outlining the utility of subscribing to data with different parameters.

### 3.1. Push

Pushing data off the box can be done synchronously at fixed intervals, or asynchronously in an ad-hoc fashion. All data pushed is set up via registered subscriptions.

#### 3.1.1. Interface counters

Interface counters provide a use case demonstrating the need to push data off of a network node at specific intervals. In this proposed framework, a node would advertise its database model including all the interfaces it has to offer and what it can count on each. A subscriber would select the interfaces and counters of each it is interested in via a filter, use the filter to group them according to available parameters, and register with the node to have them published at agreed upon intervals.

#### 3.1.2. Thresholds

Another use case demonstrating a push capability is thresholding. Assuming a node advertises the capability to record and track a threshold for a particular data type, it would use the registered subscription to push relevant data to the subscriber whenever the threshold was crossed. As an example, a subscriber may want to set a threshold for memory consumed - if the available device memory falls below a threshold the subscriber should be informed so that the operator can investigate the issue manually or programmatically.

#### 3.1.3. Streaming

Streaming data, such as RIB information, will be critical to supporting I2RS functionality. In this use case, a subscriber may desire to have all updates to a RIB streamed into the collection system, in as close to real-time as possible.

### 3.2. Pull

Pulling data off the node will always be a one-shot function. As such it is probably the most heavy-handed way to get data into the collection system, as it requires all the overhead of setting up and tearing down the control channel, exchanging the database model, creating a filter, and receiving the data. Nevertheless, it can be a valuable option and should be supported.

n.b. it is certainly possible to cache requests on publishers, and have them "replayed" via a subscription identifier. However the capability to track the state required to do so may not be available on a node, and this is somewhat counter to the overall goal of

minimizing impact to the node. Having this capability as an optional parameter of a database model, is worth exploring.

#### 3.2.1. Interface counters

Similar to the interface counter example above, except in this case the registration includes a parameter indicating the data should be collected immediately and sent only once.

#### 3.2.2. RIB Dump

Getting a snapshot of the node's current RIB can be useful for a variety of reasons. Similar to collecting RIB information above, in this example the subscriber would register for a one-shot dump of the RIB, collected and sent immediately.

#### 3.2.3. Arbitrary data collection

Once the NMS understands a node's database model, it should be able to register for one-shot collection of any subset of that database model. Given the overheads involved, this would best be restricted to one-off collection needs, such as troubleshooting, but the use case need is solid.

### 3.3. Dynamic subscriptions

This framework should support dynamic subscription capabilities with pre-existing monitoring protocols that currently require static configuration. For example, if a node's database model indicates it support IPFIX, using the standard registration process outlined above a subscriber should be able to set up a streaming IPFIX feed. BMP and the like should also be available via this mechanism.

## 4. Subscriber versus consumer

It should be noted that because overall data collection system architecture is out of scope, it is opaque to this framework whether a subscriber is also the consumer of data. In order to maximize design options, including scalability of the overall system, both options should be supported.

### 4.1. Remapping

Remapping in this context is the ability to modify a node's database model and request the modified model be used in subscriptions. While this has interesting properties, it strays far from the primary objective of getting data off of nodes as fast as possible with as little impact as possible, and thus should be considered out of scope.

## 5. Errors

Errors happen. Many classes of errors and their handling are already well-understood and don't need to be re-iterated here. There are certainly failure modes that may be unique to I2RS or this framework, however, and we should be prepared to incorporate solutions for those.

For example, providing a method for a node and a subscriber to agree on resolution steps after defined error events would be very useful. A subscriber may want certain subscriptions to be available for pulling, if the push mechanism failed.

There may also be value in defining how a subscriber can probe the transport layer, such that publisher responses can assist in troubleshooting protocol-specific failures.

The framework needs to support standardized handling of stale data. This class of error will largely be related to handling changes and exceptions in the database models exchanged. For example what happens when a node's physical configuration changes and part of an existing subscription becomes invalid. Similar thought to logical changes, such as the disappearance of a BGP speaker, needs to be given.

## 6. IANA Considerations

This document makes no request of the IANA.

## 7. Security Considerations

I2RS provides security requirements, any security requirements raised by this framework should be encompassed there.

[TODO(WK, SW): This section needs more work / text ]

## 8. Acknowledgements

The author wishes to acknowledge the contributions of a number of folk, including

{TODO(WK, SW): Remember to add folk! }



## 9. References

### 9.1. Normative References

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

### 9.2. Informative References

[DeBoer] De Boer, M. and J. Bosma, "Discovering Path MTU black holes on the Internet using RIPE Atlas", July 2012, <<http://www.nlnetlabs.nl/downloads/publications/pmtu-black-holes-msc-thesis.pdf>>.

## Authors' Addresses

Scott Whyte  
Google Inc.  
1600 Amphitheatre Parkway  
Mountain view, California 94043  
USA

Email: [swhyte@google.com](mailto:swhyte@google.com)

Marcus Hines  
Google, Inc.  
1600 Amphitheatre Parkway  
Mountain view, California 94043  
USA

Email: [hines@google.com](mailto:hines@google.com)

Warren Kumari  
Google, Inc.  
1600 Amphitheatre Parkway  
Mountain view, California 94043  
USA

Email: [warren@kumari.net](mailto:warren@kumari.net)

Network Working Group  
Internet-Draft  
Intended status: Informational  
Expires: April 24, 2014

L. Zhang  
Z. Li  
Huawei Technologies  
D. Liu  
China Mobile  
October 21, 2013

Use Cases of I2RS in Mobile Backhaul Network  
draft-zhang-i2rs-mbb-usecases-00

## Abstract

In mobile backhaul network, traditional configuration and diagnoses mechanisms base on device-level management tools and manual processing are ill-suited to meet the requirements of today's scalable, flexible, and complex network. Thanks to the new innovation of Interface to the Routing System's (I2RS) programmatic interfaces, as defined in [I-D.ward-i2rs-framework], an alternative way has been rolled out to control the configuration and diagnose the operation results. This document discusses the use case of I2RS in mobile backhaul network.

## Requirements Language

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 RFC 2119 [RFC2119].

## 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 April 24, 2014.

## Copyright Notice

Copyright (c) 2013 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

1. Introduction . . . . .	2
2. Definitions . . . . .	3
3. Application Configuration . . . . .	4
3.1. Application Configuration . . . . .	4
3.2. Requirements for I2RS . . . . .	5
4. Route Policy Enforcement . . . . .	5
4.1. Route Policy Description . . . . .	5
4.2. Requirements for I2RS . . . . .	6
5. Service Tunnel Implementation . . . . .	6
5.1. Service Tunnel Description . . . . .	6
5.2. Requirements for I2RS . . . . .	7
6. Protection Mechanism . . . . .	7
6.1. Protection Mechanism Description . . . . .	7
6.2. Requirements for I2RS . . . . .	8
7. Network Monitoring . . . . .	8
7.1. Network Monitoring Description . . . . .	8
7.2. Requirements for I2RS . . . . .	8
8. Security Considerations . . . . .	9
9. References . . . . .	9
9.1. Normative References . . . . .	9
9.2. Informative Reference . . . . .	9
Authors' Addresses . . . . .	10

## 1. Introduction

In mobile backhaul network, traditional configuration and diagnoses mechanisms base on device-level management tools and manual processing are ill-suited to meet the requirements of today's scalable, flexible, and complex network. The mobile backhaul network now need to serve various radio access modes and applications across 2G/3G / LTE/5G, build various network architectures based on the amount of network devices or the integration of different Areas or Autonomous System Numbers (ASNs), and support various network

protocols can be adopted to meet different network requirements, which make the mobile backhaul network configuration more and more arduous.

Interface to the Routing System's (I2RS) Programmatic interfaces, as defined in [I-D.ward-i2rs-framework], provides an alternative way to control the configuration and diagnose the operation results. The use cases described in this document cover the critical elements of mobile backhaul network, such as: application configuration, route policy enforcement, service tunnel implementation, protection mechanism and network monitoring. The goal is to inform the community's understanding of the mobile backhaul requirements for I2RS in a viewpoint of entire network solution.

## 2. Definitions

I2RS: Interface to the Routing System

IGP: Interior Gateway Protocol

BGP: Border Gateway Protocol

MPLS: Multi-Protocol Label Switching

LDP: Equal Cost Multi-path

RSVP-TE: Resource Reservation Protocol Traffic Engineering

PWE3: Pseudo Wire Emulation Edge-to-Edge

VPN: Virtual Private Network

L2VPN: L2 Virtual Private Network

L3VPN: L3 Virtual Private Network

SS-PW: Single Segment PW

MS-PW: Multi-Segment PW

HVPN: Hierarchical VPN

EPC: Pseudo Wire Emulation Edge-to-Edge

LTE: Long Term Evolution

FRR: Fast Reroute

ECMP: Equal Cost Multi-path

### 3. Application Configuration

#### 3.1. Application Configuration

The mobile backhaul network has evolved into an IP-based network, which faces three main challenges in network construction, including:

##### 1. various radio access modes:

Due to protect existing investment and end user resource, TDM/ATM-based access mode belongs to 2G and 3G will coexist with Ethernet-based access mode belongs to 3G, LTE and 5G for a long period in future. The radio architecture evolution will bring out new radio interfaces, such as X2 interface in LTE which will not work in hub-spoke communication mode any more while needs much more shorter time delay. A mobile backhaul network must be built to have the ability to adapt to all of the mobile access modes, providing PWE3 service for TDM /ATM-based access mode and Native IP/Ethernet, PWE3/VPLS or L3VPN service for IP-based access mode.

##### 2. various radio applications:

A variety of radio applications (such as OM, signaling, data, video, etc. ) which have different quality of services (QoS), should be delivered in specific service channels in mobile backhaul network, that means there will be more than one PW or L3VPN instances binding with specific interfaces and service tunnels.

##### 3. various network architectures:

The mobile backhaul network maybe consist of hundreds of nodes in a small county or thousands of nodes in a populous region. It will be an integration of different ASNs rather than a single AS, when the EPC is deployed in the Core network when LTE. The network devices on different points of the network (e.g. access\aggregation\core) have different routing and protocol processing capabilities, resulting in an integration of different IGP routing areas rather than a single large IGP routing area. Refer to various network architectures, different service modes should be provided, such as SS- PW or MS-PW, E2E L3VPN or HVPN, Seamless MPLS, and the integration of them.

### 3.2. Requirements for I2RS

The challenges in mobile backhaul network construction show the flexibility and complexity requirements of network configuration and modification, such as where the T-LDP should be configured, where the BGP peer should be established, where the VPN instance should be deployed, and where the BGP LSP should be set up. Faced with flat or reduced budget, the network operators are trying to squeeze the most from their network using device-level management tools and manual processing. In contrast to management of entire network devices, I2RS' programmable interface would allow network operators to distribute such configuration from a central location where a global mobile backhaul network solution provisioning information could be stored. Use of I2RS controllers to announce network configuration information would simplify and automate configuration of mobile backhaul network that readily adapt to changing network scales and radio applications.

## 4. Route Policy Enforcement

### 4.1. Route Policy Description

The route policy in mobile backhaul network mainly refers to BGP policy when L3VPN is used to serve the radio applications. The complexity of today's network architecture and radio interfaces make it very difficult to apply a network-wide route policy, for:

- o avoiding route advertisement across entire network

When a mobile backhaul network contains more than 500 nodes, complementing a multi-segments service like HVPN is recommended to reduce the routing and protocol processing quantity of network devices. BGP policy should be configured with prefix filters to advertise only the default or aggregate route to the access nodes which have poor capability, while to advertise the whole network routes to the core nodes which must have capability of fairly large routes.

- o supporting best route selection for VPN FRR or ECMP

The mobile backhaul network is recommended to be built with a multi-homed network architecture for node failure protection, where VPN FRR or ECMP should be configured. The best route selection relay on the BGP Policy using Local Preference, MED or other path attributes defined in [RFC4760]. When BGP RR is adopted to simplify the BGP peer architecture from full-mesh mode, the policy would be more complex, in some cases may be per-peer or per-route more worse.

- o allowing On-demand route advertisement

The advent of X2 interface in LTE, which needs specific route information between any two access nodes, make the network route advertisement more dynamically and unpredictably. The BGP policy should be adjusted dynamically to meet this route advertisement need across the entire network.

#### 4.2. Requirements for I2RS

Route policy enforcement in mobile backhaul network need to be much more dynamical and flexible, and when to implement a network-wide route policy, network operators should configure thousands of devices with different policy details individually according to the role of the devices, such as ASRs in one AS, ASBRs between different ASs and other service-touch nodes. It will take hours, in some cases days to configure the route policy across the entire network. I2RS controllers could use common APIs to collect network information required to create different BGP policies dynamically, then push such policy onto the corresponding network device, and make it tack effect automatically.

### 5. Service Tunnel Implementation

#### 5.1. Service Tunnel Description

In mobile backhaul network, more than one kind of Service Tunnel can be used according to network ability or other consideration in different scenarios. The Tunnel deployment use case in mobile backhaul includes:

- o MPLS LDP LSP

MPLS LDP LSP is set up through LDP protocol. Both Label Advertisement Mode of Downstream Unsolicited (DU) and Downstream on Demand (DOD) defined in [RFC3036] can be used individually or integrated across access network and aggregate/core network. If needed, the longed length match define in [RFC5283] for LDP LSP should be supported. MPLS LDP LSP has great scalability with flexible policy to control the label advertisement of route, especially in DU mode, to decrease the needless LSPs, that is help to reduce the LSP capability requirement of network devices.

- o MPLS-TE LSP

MPLS-TE LSP is set up through RSVP-TE protocol, which has multiple path control attributes (such as explicit-path, path

affinity property , path bandwidth assurance , path hop limitation, e.g.) and multiple protection modes (such as hot-standby, Fast Re-Route, protection group, e.g.). MPLS-TE LSP should be designed using the attributes and protection modes according to the requirements of the service delivery individually of integrated across access network and aggregate/core network.

- o MPLS-TP LSP

MPLS-TP includes unidirectional LSP, bidirectional co-routed LSP, and bidirectional associated LSP, which can be calculated and set up manually or using dynamic network protocols such as GMPLS. In mobile backhaul network, the LSP selection depends on the service need, and the creation of MPLS-TP LSP is always asked to be decoupled with the protocol control plane running on the separate network devices. In this case, the static MPLS-TP LSP should be designed and configured on the centralized control plane ideally.

## 5.2. Requirements for I2RS

Since mobile backhaul network is divided into access network and aggregation/core network, where service tunnel implementation is not constant and unique, perhaps need to deploy different kind of LSPs separately (such as LDP LSP or MPLS-TE LSP in both access network and aggregate/core network) or simultaneously (such as MPLS-TP static LSP in access network while LDP LSP or MPLS-TE LSP in aggregate/core network). in this case, network operators should clearly know the ability of all of the network devices and the service requirements to make the most appropriate tunnel implementation. I2RS provide a centralized control of network devices, which should automate the collection and analysis of the device ability to calculate optimal LSP path and distribute the configuration to exact devices.

## 6. Protection Mechanism

### 6.1. Protection Mechanism Description

The SLA for radio services is strict, which need interworking among multiple protection mechanisms. Two critical aspects should be taken into account for inter-working, hierarchical protection architectures and multiple OAM protocol interactions.

#### 1. tunnel protection:

The protection mechanism of different tunnel protocols, mentioned above, is different from each other. To enhance the



reliability, LDP LSP should configure LDP FRR, which is calculated depends on the protect route algorithm, which can be Loop-Free Algorithm (LFA), Remote-LFA, or Maximally Redundant Trees (MRT) used together with LDP MT described in [I-D.ietf-mpls-ldp-multi-topology]. MPLS-TE LSP should apply TE Fast Reroute or TE hot-standby. When MPLS-TP LSP is used, the LSP protection group should be configured with 1:1 or 1+1 mode for MPLS-TP line protection, as well as wrapping or steering modes fault processing for MPLS-TP ring protection.

## 2. service protection:

Service protection is commended to be configured for node failure handover in mobile backhaul network, where PW redundancy defined in [RFC6718] or BGP VPN FRR or ECMP realization should be deployed exactly.

## 6.2. Requirements for I2RS

The hierarchical protection architecture in mobile backhaul network offer high network reliability but more flexibility to meet the various needs of the tunnels and services. I2RS interface in this use case is needed to automate the configuration and make tunnel protection and service protection interwork with each other ideally.

## 7. Network Monitoring

### 7.1. Network Monitoring Description

The mobile backhaul network operators are asked to give an accurate positioning when a link or node failure occurred, and get the real reason for service quality descent. This need to apply different network monitor tools for different service mode, like Network Quality Analysis (NQA), MPLS-TP OAM, and IP Flow Performance Monitor (IPFPM). In addition, to acquire the exact traffic path is really significant when use IPFPM for point-to-point detection.

Multiple monitor tools require network operators to distinguish granular traffic flow to apply the appropriate one. At the same time, traditional device-level management tools is difficult to get the traffic path, which may need enhance the existing protocols or design a new specific protocol to do this work, both will increase the burden of mobile backhaul network.

### 7.2. Requirements for I2RS

I2RS should solve the two problems mentioned above naturally by using centralized controllers, which control and manage the entire network

devices and store the whole routing and service information directly. Meanwhile, the defect and traffic congestion or dropping can be detected real-time with I2RS, which make the network keep optimal state dynamically all the time.

## 8. Security Considerations

The mobile backhaul network use cases described in this document assumes use of I2RS's programmatic interfaces described in the I2RS framework mentioned in[I-D.ward-i2rs-framework]. This document does not change the underlying security issues inherent in the existing [I-D.ward-i2rs-framework].

## 9. References

### 9.1. Normative References

[I-D.ward-i2rs-framework]  
Atlas, A., Nadeau, T., and D. Ward, "Interface to the Routing System Framework", draft-ward-i2rs-framework-00 (work in progress), February 2013.

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

### 9.2. Informative Reference

[I-D.ietf-mpls-ldp-multi-topology]  
Zhao, Q., Fang, L., Zhou, C., Li, L., and K. Raza, "LDP Extensions for Multi Topology Routing", draft-ietf-mpls-ldp-multi-topology-09 (work in progress), October 2013.

[I-D.ietf-mpls-seamless-mpls]  
Leymann, N., Decraene, B., Filsfils, C., Konstantynowicz, M., and D. Steinberg, "Seamless MPLS Architecture", draft-ietf-mpls-seamless-mpls-04 (work in progress), July 2013.

[I-D.li-mpls-seamless-mpls-mbb]  
Li, Z., Li, L., Morillo, M., and T. Yang, "Seamless MPLS for Mobile Backhaul", draft-li-mpls-seamless-mpls-mbb-00 (work in progress), July 2013.

[RFC3036] Andersson, L., Doolan, P., Feldman, N., Fredette, A., and B. Thomas, "LDP Specification", RFC 3036, January 2001.

[RFC4760] Bates, T., Chandra, R., Katz, D., and Y. Rekhter, "Multiprotocol Extensions for BGP-4", RFC 4760, January 2007.

- [RFC5283] Decraene, B., Le Roux, J.L., and I. Minei, "LDP Extension for Inter-Area Label Switched Paths (LSPs)", RFC 5283, July 2008.
- [RFC6718] Muley, P., Aissaoui, M., and M. Bocci, "Pseudowire Redundancy", RFC 6718, August 2012.

## Authors' Addresses

Li Zhang  
Huawei Technologies  
Huawei Bld., No.156 Beiqing Rd.  
Beijing 100095  
China  
  
Email: monica.zhangli@huawei.comZ

Zhenbin Li  
Huawei Technologies  
Huawei Bld., No.156 Beiqing Rd.  
Beijing 100095  
China  
  
Email: lizhenbin@huawei.com

Dapeng Liu  
China Mobile  
Unit2, 28 Xuanwumenxi Ave,Xuanwu District  
Beijing 100053  
China  
  
Email: liudapeng@chinamobile.com