Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: April 17, 2016

A. Lindem, Ed.
Y. Qu
D. Yeung
Cisco Systems
I. Chen
Ericsson
J. Zhang
Juniper Networks
Y. Yang
Cisco Systems
October 15, 2015

Key Chain YANG Data Model
draft-acee-rtg-yang-key-chain-09.txt

Abstract

This document describes the key chain YANG data model. A key chain is a list of elements each containing a key, send lifetime, accept lifetime, and algorithm. By properly overlapping the send and accept lifetimes of multiple key chain elements, keys and algorithms may be gracefully updated. By representing them in a YANG data model, key distribution can be automated. Key chains are commonly used for routing protocol authentication and other applications. In some applications, the protocols do not use the key chain element key directly, but rather a key derivation function is used to derive a short-lived key from the key chain element key.

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 17, 2016.

Table of Contents

1. Introduction

This document describes the key chain YANG data model. A key chain is a list of elements each containing a key, send lifetime, accept lifetime, and algorithm. By properly overlapping the send and accept lifetimes of multiple key chain elements, keys and algorithms may be gracefully updated. By representing them in a YANG data model, key distribution can be automated. Key chains are commonly used for routing protocol authentication and other applications. In some applications, the protocols do not use the key chain element key directly, but rather a key derivation function is used to derive a short-lived key from the key chain element key.

1.1. Requirements Notation

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-KEYWORDS].

2. Problem Statement

This document describes a YANG [YANG] data model for key chains. Key chains have been implemented and deployed by a large percentage of network equipment vendors. Providing a standard YANG model will facilitate automated key distribution and non-disruptive key rollover. This will aid in tightening the security of the core routing infrastructure as recommended in [IAB-REPORT].

A key chain is a list containing one or more elements containing a Key ID, key, send/accept lifetimes, and the associated authentication or encryption algorithm. A key chain can be used by any service or application requiring authentication or encryption. In essence, the key-chain is a reusable key policy that can be referenced where ever it is required. The key-chain construct has been implemented by most networking vendors and deployed in many networks.

A conceptual representation of a crypto key table is described in [CRYPTO-KEYTABLE]. The crypto key table also includes keys as well as their corresponding lifetimes and algorithms. Additionally, the key table includes key selection criteria and envisions a deployment model where the details of the applications or services requiring authentication or encryption permeate into the key database. The YANG key-chain model described herein doesn’t include key selection criteria or support this deployment model. At the same time, it does not preclude it. The draft [YANG-CRYPTO-KEYTABLE] describes augmentations to the key chain YANG model in support of key selection criteria.

2.1. Graceful Key Rollover using Key Chains

Key chains may be used to gracefully update the key and/or algorithm used by an application for authentication or encryption. This MAY be accomplished by accepting all the keys that have a valid accept lifetime and sending the key with the most recent send lifetime. One scenario for facilitating key rollover is to:

1. Distribute a key chain with a new key to all the routers or other network devices in the domain of that key chain. The new key’s accept lifetime should be such that it is accepted during the key rollover period. The send lifetime should be a time in the future when it can be assured that all the routers in the domain

of that key are upgraded. This will have no immediate impact on the keys used for transmission.

2. Assure that all the network devices have been updated with the updated key chain and that their system times are roughly synchronized. The system times of devices within an administrative domain are commonly synchronized (e.g., using Network Time Protocol (NTP) [NTP-PROTO]). This also may be automated.

3. When the send lifetime of the new key becomes valid, the network devices within the domain of key chain will start sending the new key.

4. At some point in the future, a new key chain with the old key removed may be distributed to the network devices within the domain of the key chain. However, this may be deferred until the next key rollover. If this is done, the key chain will always include two keys; either the current and future key (during key rollovers) or the current and previous keys (between key rollovers).

3. Design of the Key Chain Model

The ietf-keychain module contains a list of one or more keys indexed by a Key ID. For some applications (e.g., OSPFv3 [OSPFV3-AUTH]), the Key-Id is used to identify the key chain entry to be used. In addition to the Key-ID, each key chain entry includes a key-string and a cryptographic algorithm. Optionally, the key chain entries include send/accept lifetimes. If the send/accept lifetime is unspecified, the key is always considered valid.

Note that asymmetric keys, i.e., a different key value used for transmission versus acceptance, may be supported with multiple key chain elements where the accept-lifetime or send-lifetime is not valid (e.g., has an end-time equal to the start-time).

Due to the differences in key chain implementations across various vendors, some of the data elements are optional. Additionally, the key-chain is made a grouping so that an implementation could support scoping other than at the global level. Finally, the crypto-algorithm-types grouping is provided for reuse when configuring legacy authentication and encryption not using key-chains.

A key-chain is identified by a unique name within the scope of the network device. The "key-chain-ref" typedef SHOULD be used by other YANG modules when they need to reference a configured key-chain.

3.1. Key Chain Operational State

The key chain operational state is maintained in the key-chain entries along with the configuration state. The key string itself is omitted from the operational state to minimize visibility similar to what was done with keys in SNMP MIBs. This is an area for further discussion. Additionally, the operational state includes an indication of whether or not a key chain entry is valid for sending or acceptance.

3.2. Key Chain Model Features

Features are used to handle differences between vendor implementations. For example, not all vendors support configuration an acceptance tolerance or configuration of key strings in hexadecimal. They are also used to support of security requirements (e.g., TCP-AO Algorithms [TCP-AO-ALGORITHMS]) not implemented by vendors or only a single vendor.

3.3. Key Chain Model Tree

```
+--rw key-chains
   +--rw key-chain-list* [name]
   |  +--rw name                        string
   |  +--ro name-state?                 string
   |  +--rw accept-tolerance {accept-tolerance}?
   |  |  +--rw duration?   uint32
   |  +--ro accept-tolerance-state
   |  |  +--ro duration?   uint32
   |  +--rw key-chain-entry* [key-id]
   |     +--rw key-id                      uint64
   |     +--ro key-id-state?               uint64
   |     +--rw key-string
   |     |  +--rw (key-string-style)?
   |     |     +--:(keystring)
   |     |     |  +--rw keystring?             string
   |     |     +--:(hexadecimal) {hex-key-string}?
   |     |        +--rw hexadecimal-string?   yang:hex-string
   |     +--rw lifetime
   |     |  +--rw (lifetime)?
   |     |     +--:(send-and-accept-lifetime)
   |     |     |  +--rw send-accept-lifetime
   |     |     |     +--rw (lifetime)?
   |     |     |        +--:(always)
   |     |     |        |  +--rw always?            empty
   |     |     |        +--:(start-end-time)
   |     |     |           +--rw start-date-time?
   |     |     |                   yang:date-and-time
```

```
|     |     |           +--rw (end-time)?
|     |     |              +--:(infinite)
|     |     |              |  +--rw no-end-time?       empty
|     |     |              +--:(duration)
|     |     |              |  +--rw duration?          uint32
|     |     |              +--:(end-date-time)
|     |     |                 +--rw end-date-time?
|     |     |                       yang:date-and-time
|     |     +--:(independent-send-accept-lifetime)
|     |           {independent-send-accept-lifetime}?
|     |        +--rw send-lifetime
|     |        |  +--rw (lifetime)?
|     |        |     +--:(always)
|     |        |     |  +--rw always?            empty
|     |        |     +--:(start-end-time)
|     |        |        +--rw start-date-time?
|     |        |        |     yang:date-and-time
|     |        |        +--rw (end-time)?
|     |        |           +--:(infinite)
|     |        |           |  +--rw no-end-time?       empty
|     |        |           +--:(duration)
|     |        |           |  +--rw duration?          uint32
|     |        |           +--:(end-date-time)
|     |        |              +--rw end-date-time?
|     |        |                    yang:date-and-time
|     |        +--rw accept-lifetime
|     |           +--rw (lifetime)?
|     |              +--:(always)
|     |              |  +--rw always?            empty
|     |              +--:(start-end-time)
|     |                 +--rw start-date-time?
|     |                 |       yang:date-and-time
|     |                 +--rw (end-time)?
|     |                    +--:(infinite)
|     |                    |  +--rw no-end-time?       empty
|     |                    +--:(duration)
|     |                    |  +--rw duration?          uint32
|     |                    +--:(end-date-time)
|     |                       +--rw end-date-time?
|     |                             yang:date-and-time
|     +--ro lifetime-state
|     |  +--ro send-lifetime
|     |  |  +--ro (lifetime)?
|     |  |     +--:(always)
|     |  |     |  +--ro always?            empty
|     |  |     +--:(start-end-time)
|     |  |        +--ro start-date-time?   yang:date-and-time
|     |  |        +--ro (end-time)?
```

```
|     |  |           +--:(infinite)
|     |  |           |  +--ro no-end-time?       empty
|     |  |           +--:(duration)
|     |  |           |  +--ro duration?          uint32
|     |  |           +--:(end-date-time)
|     |  |              +--ro end-date-time?
|     |  |                    yang:date-and-time
|     |  +--ro send-valid?        boolean
|     |  +--ro accept-lifetime
|     |  |  +--ro (lifetime)?
|     |  |     +--:(always)
|     |  |     |  +--ro always?            empty
|     |  |     +--:(start-end-time)
|     |  |        +--ro start-date-time?   yang:date-and-time
|     |  |        +--ro (end-time)?
|     |  |           +--:(infinite)
|     |  |           |  +--ro no-end-time?       empty
|     |  |           +--:(duration)
|     |  |           |  +--ro duration?          uint32
|     |  |           +--:(end-date-time)
|     |  |              +--ro end-date-time?
|     |  |                    yang:date-and-time
|     |  +--ro accept-valid?      boolean
|     +--rw crypto-algorithm
|     |  +--rw (algorithm)?
|     |     +--:(hmac-sha-1-12) {crypto-hmac-sha-1-12}?
|     |     |  +--rw hmac-sha1-12?       empty
|     |     +--:(aes-cmac-prf-128) {aes-cmac-prf-128}?
|     |     |  +--rw aes-cmac-prf-128?   empty
|     |     +--:(md5)
|     |     |  +--rw md5?                empty
|     |     +--:(sha-1)
|     |     |  +--rw sha-1?              empty
|     |     +--:(hmac-sha-1)
|     |     |  +--rw hmac-sha-1?         empty
|     |     +--:(hmac-sha-256)
|     |     |  +--rw hmac-sha-256?       empty
|     |     +--:(hmac-sha-384)
|     |     |  +--rw hmac-sha-384?       empty
|     |     +--:(hmac-sha-512)
|     |        +--rw hmac-sha-512?       empty
|     +--ro crypto-algorithm-state
|        +--ro (algorithm)?
|           +--:(hmac-sha-1-12) {crypto-hmac-sha-1-12}?
|           |  +--ro hmac-sha1-12?       empty
|           +--:(aes-cmac-prf-128) {aes-cmac-prf-128}?
|           |  +--ro aes-cmac-prf-128?   empty
|           +--:(md5)
```

```
      |           |  +--ro md5?               empty
      |           +--:(sha-1)
      |           |  +--ro sha-1?             empty
      |           +--:(hmac-sha-1)
      |           |  +--ro hmac-sha-1?        empty
      |           +--:(hmac-sha-256)
      |           |  +--ro hmac-sha-256?      empty
      |           +--:(hmac-sha-384)
      |           |  +--ro hmac-sha-384?      empty
      |           +--:(hmac-sha-512)
      |              +--ro hmac-sha-512?      empty
      +--rw aes-key-wrap {aes-key-wrap}?
      |  +--rw enable?   boolean
      +--ro aes-key-wrap-state {aes-key-wrap}?
         +--ro enable?   boolean
```

4. Key Chain YANG Model

```
<CODE BEGINS> file "ietf-key-chain@2015-10-15.yang"
module ietf-key-chain {
  namespace "urn:ietf:params:xml:ns:yang:ietf-key-chain";
  // replace with IANA namespace when assigned
  prefix "key-chain";

  import ietf-yang-types {
    prefix "yang";
  }

  organization
    "IETF RTG (Routing) Working Group";
  contact
    "Acee Lindem - acee@cisco.com";

  description
    "This YANG module defines the generic configuration
     data for key-chain. It is intended that the module
     will be extended by vendors to define vendor-specific
     key-chain configuration parameters.

     Copyright (c) 2015 IETF Trust and the persons identified as
     authors of the code.  All rights reserved.

     Redistribution and use in source and binary forms, with or
     without modification, is permitted pursuant to, and subject
     to the license terms contained in, the Simplified BSD License
     set forth in Section 4.c of the IETF Trust's Legal Provisions
     Relating to IETF Documents
     (http://trustee.ietf.org/license-info).
```

```
        This version of this YANG module is part of RFC XXXX; see
        the RFC itself for full legal notices.";

      revision 2015-10-15 {
        description
          "Updated version, organization, and copyright.
           Added aes-cmac-prf-128 and aes-key-wrap features.";
        reference
          "RFC XXXX: A YANG Data Model for key-chain";
      }
      revision 2015-06-29 {
        description
          "Updated version. Added Operation State following
           draft-openconfig-netmod-opstate-00.";
        reference
          "RFC XXXX: A YANG Data Model for key-chain";
      }
      revision 2015-02-24 {
        description
          "Initial revision.";
        reference
          "RFC XXXX: A YANG Data Model for key-chain";
      }

      typedef key-chain-ref {
        type leafref {
          path "/key-chain:key-chains/key-chain:key-chain-list/"
             + "key-chain:name";
        }
        description
          "This type is used by data models that need to reference
           configured key-chains.";
      }

      /* feature list */
      feature hex-key-string {
        description
          "Support hexadecimal key string.";
      }

      feature accept-tolerance {
        description
          "To specify the tolerance or acceptance limit.";
      }

      feature independent-send-accept-lifetime {
        description
          "Support for independent send and accept key lifetimes.";
```

```
  }

  feature crypto-hmac-sha-1-12 {
    description
      "Support for TCP HMAC-SHA-1 12 byte digest hack.";
  }

  feature aes-cmac-prf-128 {
    description
      "Support for AES Cipher based Message Authentication Code
       Pseudo Random Function.";
  }

  feature aes-key-wrap {
    description
      "Support for Advanced Encryption Standard (AES) Key Wrap.";
  }

  /* groupings */
  grouping lifetime {
    description
      "Key lifetime specification.";
    choice lifetime {
      default always;
      description
        "Options for specifying key accept or send lifetimes";
      case always {
        leaf always {
          type empty;
            description
              "Indicates key lifetime is always valid.";
        }
      }
      case start-end-time {
        leaf start-date-time {
          type yang:date-and-time;
          description "Start time.";
        }
        choice end-time {
          default infinite;
          description
            "End-time setting.";
          case infinite {
        leaf no-end-time {
              type empty;
            description
              "Indicates key lifetime end-time in infinite.";
            }
```

```
            }
            case duration {
              leaf duration {
                type uint32 {
                  range "1..2147483646";
                }
                units seconds;
                description "Key lifetime duration, in seconds";
              }
            }
            case end-date-time {
              leaf end-date-time {
                type yang:date-and-time;
                description "End time.";
              }
            }
          }
        }
      }
    }

  grouping crypto-algorithm-types {
      description "Cryptographic algorithm types.";
    choice algorithm {
      description
          "Options for cryptographic algorithm specification.";
      case hmac-sha-1-12 {
          if-feature crypto-hmac-sha-1-12;
          leaf hmac-sha1-12 {
              type empty;
              description "The HMAC-SHA1-12 algorithm.";
          }
      }
      case aes-cmac-prf-128 {
          if-feature aes-cmac-prf-128;
          leaf aes-cmac-prf-128 {
              type empty;
              description "The AES-CMAC-PRF-128 algorithm - required
                           by RFC 5926 for TCP-AO key derivation
                           functions.";
          }
      }
      case md5 {
          leaf md5 {
              type empty;
              description "The MD5 algorithm.";
          }
      }
```

```
        case sha-1 {
            leaf sha-1 {
                type empty;
                description "The SHA-1 algorithm.";
            }
        }
        case hmac-sha-1 {
            leaf hmac-sha-1 {
                type empty;
                description "HMAC-SHA-1 authentication algorithm.";
            }
        }
        case hmac-sha-256 {
            leaf hmac-sha-256 {
                type empty;
                description "HMAC-SHA-256 authentication algorithm.";
            }
        }
        case hmac-sha-384 {
            leaf hmac-sha-384 {
                type empty;
                description "HMAC-SHA-384 authentication algorithm.";
            }
        }
        case hmac-sha-512 {
            leaf hmac-sha-512 {
                type empty;
                description "HMAC-SHA-512 authentication algorithm.";
            }
        }
      }
    }

     grouping key-chain {
       description
         "key-chain specification grouping.";
       leaf name {
         type string;
         description "Name of the key-chain.";
       }

       leaf name-state {
         type string;
         config false;
         description "Configured name of the key-chain.";
       }

       container accept-tolerance {
```

```
          if-feature accept-tolerance;
          description
            "Tolerance for key lifetime acceptance (seconds).";
          leaf duration {
            type uint32;
            units seconds;
            default "0";
            description
              "Tolerance range, in seconds.";
          }
        }

        container accept-tolerance-state {
          config false;
          description
            "Configured tolerance for key lifetime
             acceptance (seconds).";
          leaf duration {
            type uint32;
            description
              "Configured tolerance range, in seconds.";
          }
        }

        list key-chain-entry {
          key "key-id";
          description "One key.";
          leaf key-id {
            type uint64;
            description "Key ID.";
          }
          leaf key-id-state {
            type uint64;
            config false;
            description "Configured Key ID.";
          }
          container key-string {
            description "The key string.";
            choice key-string-style {
               description
                 "Key string styles";
               case keystring {
                 leaf keystring {
                   type string;
                   description "Key string in ASCII format.";
                 }
               }
               case hexadecimal {
```

```
                if-feature hex-key-string;
                leaf hexadecimal-string {
                  type yang:hex-string;
                  description
                    "Key in hexadecimal string format.";
                }
              }
            }
          }
          container lifetime {
            description "Specify a key's lifetime.";
            choice lifetime {
              description
                "Options for specification of send and accept
                 lifetimes.";
              case send-and-accept-lifetime {
                description
                  "Send and accept key have the same lifetime.";
                container send-accept-lifetime {
                  uses lifetime;
                  description
                    "Single lifetime specification for both send and
                     accept lifetimes.";
                }
              }
              case independent-send-accept-lifetime {
                if-feature independent-send-accept-lifetime;
                description
                  "Independent send and accept key lifetimes.";
                container send-lifetime {
                  uses lifetime;
                  description
                    "Separate lifetime specification for send
                     lifetime.";
                }
                container accept-lifetime {
                  uses lifetime;
                  description
                    "Separate lifetime specification for accept
                     lifetime.";
                }
              }
            }
          }
          container lifetime-state {
            config false;
            description "Configured key's lifetime.";
            container send-lifetime {
```

```
            uses lifetime;
            description
              "Configured send-lifetime.";
          }
          leaf send-valid {
            type boolean;
            description
              "Status of send-lifetime.";
          }
          container accept-lifetime {
            uses lifetime;
            description
              "Configured accept-lifetime.";
          }
          leaf accept-valid {
            type boolean;
            description
              "Status of accept-lifetime.";
          }
        }
        container crypto-algorithm {
          uses crypto-algorithm-types;
          description "Cryptographic algorithm associated with key.";
        }
        container crypto-algorithm-state {
          config false;
          uses crypto-algorithm-types;
          description "Configured cryptographic algorithm.";
        }
      }
    }

    container key-chains {
      list key-chain-list {
        key "name";
        description
          "List of key-chains.";
        uses key-chain;
      }
      container aes-key-wrap {
        if-feature aes-key-wrap;
        leaf enable {
          type boolean;
          default false;
          description
            "Enable AES Key Wrap encryption.";
        }
        description
```

```
            "AES Key Wrap password encryption.";
        }
        container aes-key-wrap-state {
          if-feature aes-key-wrap;
          config false;
          leaf enable {
            type boolean;
            description "AES Key Wrap state.";
          }
          description "Status of AES Key Wrap.";
        }
        description "All configured key-chains for the device.";
      }
    }
    <CODE ENDS>
```

5. Relationship to other Work

6. Security Considerations

This document enables the automated distribution of industry standard key chains using the NETCONF [NETCONF] protocol. As such, the security considerations for the NETCONF protocol are applicable. Given that the key chains themselves are sensitive data, it is RECOMMENDED that the NETCONF communication channel be encrypted. One way to do accomplish this would be to invoke and run NETCONF over SSH as described in [NETCONF-SSH].

When configured, the key-strings can be encrypted using the AES Key Wrap algorithm [AES-KEY-WRAP]. The AES key-encryption key (KEK) is not included in the YANG model and must be set or derived independent of key-chain configuration.

The key strings are not included in the operational state. This is a practice carried over from SNMP MIB modules and is an area for further discussion.

7. IANA Considerations

This document registers a URI in the IETF XML registry [XML-REGISTRY]. Following the format in RFC 3688, the following registration is requested to be made:

URI: urn:ietf:params:xml:ns:yang:ietf-key-chain

Registrant Contact: The IESG.

XML: N/A, the requested URI is an XML namespace.

This document registers a YANG module in the YANG Module Names registry [YANG].

name: ietf-acl namespace: urn:ietf:params:xml:ns:yang:ietf-key-chain prefix: ietf-key-chain reference: RFC XXXX

8. References

8.1. Normative References

[NETCONF] Enns, R., Bjorklund, M., Schoenwaelder, J., and A. Bierman, "Network Configuration Protocol (NETCONF)", RFC 6241, June 2011.

[NETCONF-SSH] Wasserman, M., "Using NETCONF Protocol over Secure Shell (SSH)", RFC 6242, June 2011.

[RFC-KEYWORDS] Bradner, S., "Key words for use in RFC's to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[XML-REGISTRY] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, January 2004.

[YANG] Bjorklund, M., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, October 2010.

8.2. Informative References

[AES-KEY-WRAP] Housley, R. and M. Dworkin, "Advanced Encryption Standard (AES) Key Wrap with Padding Algorithm", RFC 5649, August 2009.

[CRYPTO-KEYTABLE] Housley, R., Polk, T., Hartman, S., and D. Zhang, "Table of Cryptographic Keys", RFC 7210, April 2014.

[IAB-REPORT] Andersson, L., Davies, E., and L. Zhang, "Report from the IAB workshop on Unwanted Traffic March 9-10, 2006", RFC 4948, August 2007.

[NTP-PROTO]
Mills, D., Martin, J., Burbank, J., and W. Kasch, "Network Time Protocol Version 4: Protocol and Algorithms Specification", RFC 5905, June 2010.

[OSPFV3-AUTH]
Bhatia, M., Manral, V., and A. Lindem, "Supporting Authentication Trailer for OSPFv3", RFC 7166, March 2014.

[TCP-AO-ALGORITHMS]
Lebovitz, G. and E. Rescorla, "Cryptographic Algorithms for the TCP Authentication Option (TCP-AO)", draft-chen-rtg-key-table-yang-00.txt (work in progress), June 2010.

[YANG-CRYPTO-KEYTABLE]
Chen, I., "YANG Data Model for RFC 7210 Key Table", draft-chen-rtg-key-table-yang-00.txt (work in progress), March 2015.

Appendix A. Acknowledgments

The RFC text was produced using Marshall Rose’s xml2rfc tool.

Thanks to Brian Weis for fruitful discussions on security requirements.

Authors’ Addresses

Acee Lindem (editor)
Cisco Systems
301 Midenhall Way
Cary, NC 27513
USA

Email: acee@cisco.com

Yingzhen Qu
Cisco Systems
170 West Tasman Drive
San Jose, CA 95134
USA

Email: yiqu@cisco.com

Derek Yeung
Cisco Systems
170 West Tasman Drive
San Jose, CA 95134
USA

Email: myeung@cisco.com

Ing-Wher Chen
Ericsson

Email: ing-wher.chen@ericsson.com

Jeffrey Zhang
Juniper Networks
10 Technology Park Drive
Westford, MA 01886
USA

Email: zzhang@juniper.net

Yi Yang
Cisco Systems
7025 Kit Creek Road
Research Triangle Park, NC 27709
USA

Email: yiya@cisco.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 5, 2019

A. Lindem
Cisco Systems
Y. Qu
Huawei

March 4, 2019

RIB YANG Data Model
draft-acee-rtgwg-yang-rib-extend-10.txt

Abstract

The Routing Information Base (RIB) is a list of routes and their corresponding administrative data and operational state.

RFC 8349 defines the basic building blocks for RIB, and this model augments it to support multiple next-hops (aka, paths) for each route as well as additional attributes.

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 https://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 September 5, 2019.

Table of Contents

1. Introduction

This document defines a YANG, [RFC6020][RFC7950], data model which extends the generic data model for RIB by augmenting the ietf-routing model as defined in [RFC8349].

RIB is a collection of best routes from all routing protocols. Within a protocol routes are selected based on the metrics in use by that protocol, and the protocol install its best routes to RIB. RIB selects the best route by comparing the route preference (aka, administrative distance) of the associated protocol.

The augmentations described herein extend the RIB to support multiple paths per route, route metrics, and administrative tags.

The YANG modules in this document conform to the Network Management Datastore Architecture (NMDA) [RFC8342].

2. Terminology and Notation

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

The following terms are defined in [RFC8342]:

- client
- server
- configuration
- system state
- operational state
- intended configuration

The following terms are defined in [RFC7950]:

- action
- augment
- container
- container with presence
- data model
- data node
- feature
- leaf
- list
- mandatory node
- module
- schema tree

- RPC (Remote Procedure Call) operation

2.1. Glossary of New Terms

Routing Information Base (RIB): An object containing a list of routes, together with other information. See [RFC8349] Section 5.2 for details.

2.2. Tree Diagrams

Tree diagrams used in this document follow the notation defined in [RFC8340].

2.3. Prefixes in Data Node Names

In this document, names of data nodes, actions, and other data model objects are often used without a prefix, as long as it is clear from the context in which YANG module each name is defined. Otherwise, names are prefixed using the standard prefix associated with the corresponding YANG module, as shown in Table 1.

Prefix	YANG module	Reference
if	ietf-interfaces	[RFC8343]
rt	ietf-routing	[RFC8349]
v4ur	ietf-ipv4-unicast-routing	[RFC8349]
v6ur	ietf-ipv6-unicast-routing	[RFC8349]
inet	ietf-inet-types	[RFC6991]

Table 1: Prefixes and Corresponding YANG Modules

3. Design of the Model

The YANG definitions in this document augment the ietf-routing model defined in [RFC8349], which provides a basis for routing system data model development. Together with modules defined in [RFC8349], a generic RIB Yang model is defined to implement and monitor RIB.

The models in [RFC8349] also define the basic configuration and operational state for both IPv4 and IPv6 static routes and this document also provides augmentations for static routes to support multiple next-hop and more next-hop attributes.

3.1. RIB Tags and Preference

Individual routes tags will be supported at both the route and next-hop level. A preference per next-hop is also supported for selection of the most preferred reachable static route.

3.2. Multiple next-hops

Both Ipv4 and IPv6 static route configuration defined in [RFC8349] have been augmented with a multi-next-hop option.

A static route/prefix can be configured to have multiple next-hops, each with their own tag and route preference.

In RIB, a route may have multiple next-hops. They can be either equal cost multiple paths (ECMP), or they may have different metrics.

3.3. Repair path

The loop-free alternate (LFA) Fast Reroute (FRR) pre-computes repair paths by routing protocols, and RIB stores the best repair path.

A repair path is augmented in RIB operation state for each path.

4. RIB Model Tree

The tree associated with the "ietf-rib-extension" module follows. The meaning of the symbols can be found in [RFC8340]. The ietf-routing.yang tree with the augmentations herein is included in Appendix A.

```
augment /rt:routing/rt:control-plane-protocols
         /rt:control-plane-protocol/rt:static-routes/v4ur:ipv4
         /v4ur:route/v4ur:next-hop/v4ur:next-hop-options
         /v4ur:simple-next-hop:
 +--rw preference?        uint32
 +--rw tag?               uint32
 +--rw application-tag?   uint32
augment /rt:routing/rt:control-plane-protocols
         /rt:control-plane-protocol/rt:static-routes/v4ur:ipv4
         /v4ur:route/v4ur:next-hop/v4ur:next-hop-options
         /v4ur:next-hop-list/v4ur:next-hop-list/v4ur:next-hop:
 +--rw preference?        uint32
 +--rw tag?               uint32
 +--rw application-tag?   uint32
augment /rt:routing/rt:control-plane-protocols
         /rt:control-plane-protocol/rt:static-routes/v6ur:ipv6
         /v6ur:route/v6ur:next-hop/v6ur:next-hop-options
```

```
          /v6ur:simple-next-hop:
    +--rw preference?        uint32
    +--rw tag?               uint32
    +--rw application-tag?   uint32
   augment /rt:routing/rt:control-plane-protocols
            /rt:control-plane-protocol/rt:static-routes/v6ur:ipv6
            /v6ur:route/v6ur:next-hop/v6ur:next-hop-options
            /v6ur:next-hop-list/v6ur:next-hop-list/v6ur:next-hop:
    +--rw preference?        uint32
    +--rw tag?               uint32
    +--rw application-tag?   uint32
   augment /rt:routing/rt:ribs/rt:rib:
    +--ro rib-summary-statistics
       +--ro total-routes?              uint32
       +--ro total-active-routes?       uint32
       +--ro total-route-memory?        uint64
       +--ro protocol-rib-statistics* []
          +--ro rib-protocol?             identityref
          +--ro protocol-total-routes?    uint32
          +--ro protocol-active-routes?   uint32
          +--ro protocol-route-memory?    uint64
   augment /rt:routing/rt:ribs/rt:rib/rt:routes/rt:route:
    +--ro metric?            uint32
    +--ro tag?               uint32
    +--ro application-tag?   uint32
   augment /rt:routing/rt:ribs/rt:rib/rt:routes:
    +--ro repair-route* [id]
       +--ro id          string
       +--ro next-hop
       |  +--ro outgoing-interface?   if:interface-state-ref
       |  +--ro next-hop-address?     inet:ip-address
       +--ro metric?     uint32
   augment /rt:routing/rt:ribs/rt:rib/rt:routes/rt:route
            /rt:next-hop/rt:next-hop-options/rt:simple-next-hop:
    +--ro repair-path?
            -> /rt:routing/ribs/rib/routes/repair-route/id
   augment /rt:routing/rt:ribs/rt:rib/rt:routes/rt:route
            /rt:next-hop/rt:next-hop-options/rt:special-next-hop:
    +--ro repair-path?
            -> /rt:routing/ribs/rib/routes/repair-route/id
   augment /rt:routing/rt:ribs/rt:rib/rt:routes/rt:route
            /rt:next-hop/rt:next-hop-options/rt:next-hop-list
            /rt:next-hop-list/rt:next-hop:
    +--ro repair-path?
            -> /rt:routing/ribs/rib/routes/repair-route/id
```

5. RIB YANG Model

```
<CODE BEGINS> file "ietf-rib-extension@2019-03-01.yang"
module ietf-rib-extension {
  yang-version "1.1";
  namespace "urn:ietf:params:xml:ns:yang:ietf-rib-extension";

  prefix rib-ext;

  import ietf-inet-types {
    prefix "inet";
  }

  import ietf-interfaces {
    prefix "if";
  }

  import ietf-routing {
    prefix "rt";
  }

  import ietf-ipv4-unicast-routing {
    prefix "v4ur";
  }

  import ietf-ipv6-unicast-routing {
    prefix "v6ur";
  }

  organization
      "IETF RTGWG - Routing Working Group";

  contact
    "WG Web:   <http://datatracker.ietf.org/group/rtgwg/>
     WG List:  <mailto:rtgwg@ietf.org>

     Author:   Acee Lindem
               <mailto:acee@cisco.com>
     Author:   Yingzhen Qu
               <mailto:yingzhen.qu@huawei.com>";

  description
    "This YANG module extends the generic data model for
     RIB by augmenting the ietf-netmod-routing-cfg
     model. It is intended that the module will be extended
     by vendors to define vendor-specific RIB parameters.

     This YANG model conforms to the Network Management
```

```
        Datastore Architecture (NDMA) as described in RFC 8342.

        Copyright (c) 2018 IETF Trust and the persons identified as
        authors of the code.  All rights reserved.

        Redistribution and use in source and binary forms, with or
        without modification, is permitted pursuant to, and subject
        to the license terms contained in, the Simplified BSD License
        set forth in Section 4.c of the IETF Trust's Legal Provisions
        Relating to IETF Documents
        (http://trustee.ietf.org/license-info).

        This version of this YANG module is part of RFC XXXX;
        see the RFC itself for full legal notices.";

    revision 2019-03-01 {
      description
        "Initial RFC Version";
      reference
        "RFC XXXX: A YANG Data Model for RIB Extensions.";
    }

    /* Groupings */
    grouping rib-statistics {
      description "Statistics grouping used for RIB augmentation";
      container rib-summary-statistics {
        config false;
        description "Container for RIB statistics";
        leaf total-routes {
          type uint32;
          description
            "Total routes in the RIB from all protocols";
        }
        leaf total-active-routes {
          type uint32;
          description
            "Total active routes in the RIB";
        }
        leaf total-route-memory {
          type uint64;
          description
            "Total memory for all routes in the RIB from all
             protocol clients";
        }
        list protocol-rib-statistics {
          description "List protocol statistics";
          leaf rib-protocol {
            type identityref {
```

```
            base rt:routing-protocol;
          }
          description "Routing protocol for statistics";
        }
        leaf protocol-total-routes {
          type uint32;
          description
            "Total number routes for protocol in the RIB";
        }
        leaf protocol-active-routes {
          type uint32;
          description
            "Number active routes for protocol in the RIB";
        }
        leaf protocol-route-memory {
          type uint64;
          description
            "Total memory for all routes in the RIB for protocol";
        }
      }
    }
  }

  grouping next-hop {
    description
      "Next-hop grouping";
    leaf interface {
      type if:interface-ref;
      description
        "Outgoing interface";
    }
    leaf address {
      type inet:ip-address;
      description
        "IPv4 or IPv6 Address of the next-hop";
    }
  }

  grouping attributes {
    description
      "Common attributes applicable to all paths";
    leaf metric {
      type uint32;
      description "Route metric";
    }
    leaf tag {
      type uint32;
      description "Route tag";
```

```
      }
      leaf application-tag {
        type uint32;
        description "Additional Application-Specific Route tag";
      }
    }
    grouping path-attribute {
      description
        "Path attribute grouping";
      leaf repair-path {
        type leafref {
          path "/rt:routing/rt:ribs/rt:rib/"
               + "rt:routes/repair-route/id";
        }
        description
          "IP Fast ReRoute (IPFRR) repair path, use a path
           from repair-route list";
      }
    }

    augment "/rt:routing/rt:control-plane-protocols/"
          + "rt:control-plane-protocol/rt:static-routes/v4ur:ipv4/"
          + "v4ur:route/v4ur:next-hop/v4ur:next-hop-options/"
          + "v4ur:simple-next-hop"
    {
      description
        "Augment 'simple-next-hop' case in IPv4 unicast route.";
      leaf preference {
        type uint32;
        default "1";
        description "Route preference - Used to select among multiple
                     static routes with a lower preference next-hop
                     preferred and equal preference paths yielding
                     Equal Cost Multi-Path (ECMP).";
      }
      leaf tag {
        type uint32;
        default "0";
        description "Route tag";
      }
      leaf application-tag {
        type uint32;
        description "Additional Application-Specific Route tag";
      }
    }

    augment "/rt:routing/rt:control-plane-protocols/"
          + "rt:control-plane-protocol/rt:static-routes/v4ur:ipv4/"
```

```
          + "v4ur:route/v4ur:next-hop/v4ur:next-hop-options/"
          + "v4ur:next-hop-list/v4ur:next-hop-list/v4ur:next-hop"
    {
      description
        "Augment static route configuration 'next-hop-list'.";

      leaf preference {
        type uint32;
        default "1";
        description "Route preference - Used to select among multiple
                     static routes with a lower preference next-hop
                     preferred and equal preference paths yielding
                     Equal Cost Multi-Path (ECMP).";
      }
      leaf tag {
        type uint32;
        default "0";
        description "Route tag";
      }
      leaf application-tag {
        type uint32;
        description "Additional Application-Specific Route tag";
      }
  }

    augment "/rt:routing/rt:control-plane-protocols/"
          + "rt:control-plane-protocol/rt:static-routes/v6ur:ipv6/"
          + "v6ur:route/v6ur:next-hop/v6ur:next-hop-options/"
          + "v6ur:simple-next-hop"
    {
      description
        "Augment 'simple-next-hop' case in IPv6 unicast route.";
      leaf preference {
        type uint32;
        default "1";
        description "Route preference - Used to select among multiple
                     static routes with a lower preference next-hop
                     preferred and equal preference paths yielding
                     Equal Cost Multi-Path (ECMP).";
      }
      leaf tag {
        type uint32;
        default "0";
        description "Route tag";
      }
      leaf application-tag {
        type uint32;
        description "Additional Application-Specific Route tag";
```

```
      }
    }

    augment "/rt:routing/rt:control-plane-protocols/"
          + "rt:control-plane-protocol/rt:static-routes/v6ur:ipv6/"
          + "v6ur:route/v6ur:next-hop/v6ur:next-hop-options/"
          + "v6ur:next-hop-list/v6ur:next-hop-list/v6ur:next-hop"
    {
      description
        "Augment static route configuration 'next-hop-list'.";

      leaf preference {
        type uint32;
        default "1";
        description "Route preference - Used to select among multiple
                     static routes with a lower preference next-hop
                     preferred and equal preference paths yielding
                     Equal Cost Multi-Path (ECMP).";
      }
      leaf tag {
        type uint32;
        default "0";
        description "Route tag";
      }
      leaf application-tag {
        type uint32;
        description "Additional Application-Specific Route tag";
      }
    }

    augment "/rt:routing/rt:ribs/rt:rib"
    {
      description "Augment a RIB with statistics";
      uses rib-statistics;
    }

    augment "/rt:routing/rt:ribs/rt:rib/"
          + "rt:routes/rt:route"
    {
      description
        "Augment a route in RIB with tag.";
      uses attributes;
    }

    augment "/rt:routing/rt:ribs/rt:rib/"
          + "rt:routes"
    {
      description
```

```
        "Augment a route with a list of repair-paths.";
      list repair-route {
        key "id";
        description
          "A repair-path entry, which can be referenced
           by a repair-path.";
        leaf id {
          type string;
          description
            "A unique identifier.";
        }

        container next-hop {
                description
                  "Route's next-hop attribute.";
          leaf outgoing-interface {
            type if:interface-state-ref;
            description
              "Name of the outgoing interface.";
          }
          leaf next-hop-address {
            type inet:ip-address;
            description
              "IP address of the next hop.";
          }
        }
        leaf metric {
          type uint32;
          description "Route metric";
        }
      }
    }

    augment "/rt:routing/rt:ribs/rt:rib/"
          + "rt:routes/rt:route/rt:next-hop/rt:next-hop-options/"
          + "rt:simple-next-hop"
    {
       description
         "Add more parameters to a path.";
       uses path-attribute;
    }

    augment "/rt:routing/rt:ribs/rt:rib/"
          + "rt:routes/rt:route/rt:next-hop/rt:next-hop-options/"
          + "rt:special-next-hop"
    {
      description
        "Add more parameters to a path.";
```

```
      uses path-attribute;
    }

    augment "/rt:routing/rt:ribs/rt:rib/"
          + "rt:routes/rt:route/rt:next-hop/rt:next-hop-options/"
          + "rt:next-hop-list/rt:next-hop-list/rt:next-hop"
    {
      description
        "This case augments the 'next-hop-options' in the routing
         model.";
      uses path-attribute;
    }
  }
  <CODE ENDS>
```

6. Security Considerations

The YANG modules specified in this document define a schema for data that is designed to be accessed via network management protocols such as NETCONF [RFC6241] or RESTCONF [RFC8040]. The lowest NETCONF layer is the secure transport layer, and the mandatory-to-implement secure transport is Secure Shell (SSH) [RFC6242]. The lowest RESTCONF layer is HTTPS, and the mandatory-to-implement secure transport is TLS [RFC5246].

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

There are a number of data nodes defined in ietf-rib-extensions.yang module that are writable/creatable/deletable (i.e., config true, which is the default). These data nodes may be considered sensitive or vulnerable in some network environments. Write operations (e.g., edit-config) to these data nodes without proper protection can have a negative effect on network operations. For these augmentations to ietf-routing.yang, the abiliy to delete, add, and modify IPv4 and IPv6 static routes would allow traffic to be misrouted.

Some of the readable data nodes in the ietf-rib-extensions.yang module may be considered sensitive or vulnerable in some network environments. It is thus important to control read access (e.g., via get, get-config, or notification) to these data nodes. The exposure of the Routing Information Base (RIB) will expose the routing topology of the network. This may be undesirable since both due to the fact that exposure may facilitate other attacks. Additionally, network operators may consider their topologies to be sensitive confidential data.

All the security considerations for [RFC8349] writable and readable data nodes apply to the augmentations described herein.

7. IANA Considerations

This document registers a URI in the IETF XML registry [XML-REGISTRY]. Following the format in [RFC3688], the following registration is requested to be made:

URI: urn:ietf:params:xml:ns:yang:ietf-rib-extension

Registrant Contact: The IESG.

XML: N/A, the requested URI is an XML namespace.

This document registers a YANG module in the YANG Module Names registry [RFC6020].

name: ietf-acl namespace: urn:ietf:params:xml:ns:yang:ietf-rib-extension prefix: ietf-rib-ext reference: RFC XXXX

8. References

8.1. Normative References

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

[RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <https://www.rfc-editor.org/info/rfc3688>.

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

[RFC6020] Bjorklund, M., Ed., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, DOI 10.17487/RFC6020, October 2010, <https://www.rfc-editor.org/info/rfc6020>.

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

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

[RFC6991] Schoenwaelder, J., Ed., "Common YANG Data Types", RFC 6991, DOI 10.17487/RFC6991, July 2013, <https://www.rfc-editor.org/info/rfc6991>.

[RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <https://www.rfc-editor.org/info/rfc7950>.

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

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

[RFC8341] Bierman, A. and M. Bjorklund, "Network Configuration Access Control Model", STD 91, RFC 8341, DOI 10.17487/RFC8341, March 2018, <https://www.rfc-editor.org/info/rfc8341>.

[RFC8342] Bjorklund, M., Schoenwaelder, J., Shafer, P., Watsen, K., and R. Wilton, "Network Management Datastore Architecture (NMDA)", RFC 8342, DOI 10.17487/RFC8342, March 2018, <https://www.rfc-editor.org/info/rfc8342>.

[RFC8343] Bjorklund, M., "A YANG Data Model for Interface Management", RFC 8343, DOI 10.17487/RFC8343, March 2018, <https://www.rfc-editor.org/info/rfc8343>.

[RFC8349] Lhotka, L., Lindem, A., and Y. Qu, "A YANG Data Model for Routing Management (NMDA Version)", RFC 8349, DOI 10.17487/RFC8349, March 2018, <https://www.rfc-editor.org/info/rfc8349>.

8.2. Informative References

[RFC8340] Bjorklund, M. and L. Berger, Ed., "YANG Tree Diagrams", BCP 215, RFC 8340, DOI 10.17487/RFC8340, March 2018, <https://www.rfc-editor.org/info/rfc8340>.

[XML-REGISTRY] Mealling, M., "The IETF XML Registry", BCP 81, January 2004.

Appendix A. Combined Tree Diagram

This appendix includes the combined ietf-routing.yang and ietf-rib-extensions.yang tree diagram.

```
module: ietf-routing
+--rw routing
|  +--rw router-id?                 yang:dotted-quad
|  +--ro interfaces
|  |  +--ro interface*   if:interface-ref
|  +--rw control-plane-protocols
|  |  +--rw control-plane-protocol* [type name]
|  |     +--rw type             identityref
|  |     +--rw name             string
|  |     +--rw description?     string
|  |     +--rw static-routes
|  +--rw ribs
|     +--rw rib* [name]
|        +--rw name               string
|        +--rw address-family    identityref
|        +--ro default-rib?      boolean {multiple-ribs}?
|        +--ro routes
|        |  +--ro route* []
|        |     +--ro route-preference?   route-preference
|        |     +--ro next-hop
|        |     |  +--ro (next-hop-options)
|        |     |     +--:(simple-next-hop)
|        |     |     |  +--ro outgoing-interface?   if:interface-ref
|        |     |     +--:(special-next-hop)
|        |     |     |  +--ro special-next-hop?     enumeration
|        |     |     +--:(next-hop-list)
|        |     |        +--ro next-hop-list
|        |     |           +--ro next-hop* []
|        |     |              +--ro outgoing-interface?
|        |     |                      if:interface-ref
|        |     +--ro source-protocol     identityref
|        |     +--ro active?             empty
|        |     +--ro last-updated?       yang:date-and-time
|        +---x active-route
|        |  +--ro output
|        |     +--ro route
|        |        +--ro next-hop
|        |        |  +--ro (next-hop-options)
|        |        |     +--:(simple-next-hop)
|        |        |     |  +--ro outgoing-interface?
|        |        |     |          if:interface-ref
|        |        |     +--:(special-next-hop)
|        |        |     |  +--ro special-next-hop?     enumeration
```

```
|         |         |     +--:(next-hop-list)
|         |         |        +--ro next-hop-list
|         |         |           +--ro next-hop* []
|         |         |              +--ro outgoing-interface?
|         |         |                          if:interface-ref
|         |         +--ro source-protocol    identityref
|         |         +--ro active?            empty
|         |         +--ro last-updated?      yang:date-and-time
|         +--rw description?      string
o--ro routing-state
   +--ro router-id?                  yang:dotted-quad
   o--ro interfaces
   |  o--ro interface*   if:interface-state-ref
   o--ro control-plane-protocols
   |  o--ro control-plane-protocol* [type name]
   |     o--ro type    identityref
   |     o--ro name    string
   o--ro ribs
      o--ro rib* [name]
         o--ro name               string
         +--ro address-family     identityref
         o--ro default-rib?       boolean {multiple-ribs}?
         o--ro routes
         |  o--ro route* []
         |     o--ro route-preference?   route-preference
         |     o--ro next-hop
         |     |  +--ro (next-hop-options)
         |     |     +--:(simple-next-hop)
         |     |     |  +--ro outgoing-interface?   if:interface-ref
         |     |     +--:(special-next-hop)
         |     |     |  +--ro special-next-hop?     enumeration
         |     |     +--:(next-hop-list)
         |     |        +--ro next-hop-list
         |     |           +--ro next-hop* []
         |     |              +--ro outgoing-interface?
         |     |                          if:interface-ref
         |     +--ro source-protocol     identityref
         |     +--ro active?             empty
         |     +--ro last-updated?       yang:date-and-time
         o---x active-route
            +--ro output
               o--ro route
                  o--ro next-hop
                  |  +--ro (next-hop-options)
                  |     +--:(simple-next-hop)
                  |     |  +--ro outgoing-interface?
                  |     |            if:interface-ref
                  |     +--:(special-next-hop)
```

```
                     |      |  +--ro special-next-hop?     enumeration
                     |      +--:(next-hop-list)
                     |         +--ro next-hop-list
                     |            +--ro next-hop* []
                     |               +--ro outgoing-interface?
                     |                       if:interface-ref
                     +--ro source-protocol    identityref
                     +--ro active?            empty
                     +--ro last-updated?      yang:date-and-time

   module: ietf-rib-extension
   augment /rt:routing/rt:control-plane-protocols
             /rt:control-plane-protocol/rt:static-routes/v4ur:ipv4
             /v4ur:route/v4ur:next-hop/v4ur:next-hop-options
             /v4ur:simple-next-hop:
     +--rw preference?        uint32
     +--rw tag?               uint32
     +--rw application-tag?   uint32
   augment /rt:routing/rt:control-plane-protocols
             /rt:control-plane-protocol/rt:static-routes/v4ur:ipv4
             /v4ur:route/v4ur:next-hop/v4ur:next-hop-options
             /v4ur:next-hop-list/v4ur:next-hop-list/v4ur:next-hop:
     +--rw preference?        uint32
     +--rw tag?               uint32
     +--rw application-tag?   uint32
   augment /rt:routing/rt:control-plane-protocols
             /rt:control-plane-protocol/rt:static-routes/v6ur:ipv6
             /v6ur:route/v6ur:next-hop/v6ur:next-hop-options
             /v6ur:simple-next-hop:
     +--rw preference?        uint32
     +--rw tag?               uint32
     +--rw application-tag?   uint32
   augment /rt:routing/rt:control-plane-protocols
             /rt:control-plane-protocol/rt:static-routes/v6ur:ipv6
             /v6ur:route/v6ur:next-hop/v6ur:next-hop-options
             /v6ur:next-hop-list/v6ur:next-hop-list/v6ur:next-hop:
     +--rw preference?        uint32
     +--rw tag?               uint32
     +--rw application-tag?   uint32
   augment /rt:routing/rt:ribs/rt:rib:
     +--ro rib-summary-statistics
        +--ro total-routes?              uint32
        +--ro total-active-routes?       uint32
        +--ro total-route-memory?        uint64
        +--ro protocol-rib-statistics* []
           +--ro rib-protocol?             identityref
           +--ro protocol-total-routes?    uint32
           +--ro protocol-active-routes?   uint32
```

```
         +--ro protocol-route-memory?    uint64
   augment /rt:routing/rt:ribs/rt:rib/rt:routes/rt:route:
     +--ro metric?            uint32
     +--ro tag?               uint32
     +--ro application-tag?   uint32
   augment /rt:routing/rt:ribs/rt:rib/rt:routes:
     +--ro repair-route* [id]
        +--ro id          string
        +--ro next-hop
        |  +--ro outgoing-interface?   if:interface-state-ref
        |  +--ro next-hop-address?     inet:ip-address
        +--ro metric?     uint32
   augment /rt:routing/rt:ribs/rt:rib/rt:routes/rt:route/rt:next-hop
             /rt:next-hop-options/rt:simple-next-hop:
     +--ro repair-path?   -> /rt:routing/ribs/rib/routes/repair-route/id
   augment /rt:routing/rt:ribs/rt:rib/rt:routes/rt:route/rt:next-hop
             /rt:next-hop-options/rt:special-next-hop:
     +--ro repair-path?   -> /rt:routing/ribs/rib/routes/repair-route/id
   augment /rt:routing/rt:ribs/rt:rib/rt:routes/rt:route/rt:next-hop
             /rt:next-hop-options/rt:next-hop-list/rt:next-hop-list
             /rt:next-hop:
     +--ro repair-path?   -> /rt:routing/ribs/rib/routes/repair-route/id
```

Appendix B. ietf-rib-extension.yang examples

Examples will be added in a future version of this document.

Appendix C. Acknowledgments

The RFC text was produced using Marshall Rose's xml2rfc tool.

The authors wish to thank Les Ginsberg, Krishna Deevi, and Suyoung Yoon for their helpful comments and suggestions.

The authors wish to thank Tom Petch and Rob Wilton for review and comments.

Authors' Addresses

Acee Lindem
Cisco Systems
301 Midenhall Way
Cary, NC 27513
USA

EMail: acee@cisco.com

Yingzhen Qu
Huawei
2330 Central Expressway
Santa Clara, CA 95050
USA

EMail: yingzhen.qu@huawei.com

Routing Area Working Group
Internet-Draft
Intended status: Standards Track
Expires: March 2, 2017

Anil Kumar S N
Gaurav Agrawal
Vinod Kumar S
Huawei Technologies
C. Bowers
Juniper Networks

August 29, 2016

Maximally Redundant Trees in Segment Routing
draft-agv-rtgwg-spring-segment-routing-mrt-03

Abstract

[RFC7812] defines an architecture for IP/LDP Fast Reroute using Maximally Redundant Trees (MRT-FRR). This document extends the use of MRT to provide link and node protection for networks that use segment routing. This document makes use of the inherent support in segment routing for associating segments with different algorithms for computing next hops to reach prefixes. It assigns new Segment Routing Algorithms values corresponding to the MRT-Red and MRT-Blue next-hop computations defined in [RFC7811].

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 March 2, 2017.

Table of Contents

1. Introduction

MRT is a fast re-route technology that provides 100% coverage for link and nodes failures. This document describes how to use MRT as a FRR technology in a segment routing network.

2. 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]

3. Terminology

Redundant Trees (RT): A pair of trees where the path from any node X to the root R along the first tree is node-disjoint with the path from the same node X to the root along the second tree. These can be computed in 2-connected graphs.

Maximally Redundant Trees (MRT): A pair of trees where the path from any node X to the root R along the first tree and the path from the same node X to the root along the second tree share the minimum number of nodes and the minimum number of links. Each such shared node is a cut-vertex. Any shared links are cut-links. Any RT is an MRT but many MRTs are not RTs. The two MRTs are referred to as MRT-Blue and MRT-Red.

MRT-Red: MRT-Red is used to describe one of the two MRTs; it is used to described the associated forwarding topology and MT-ID. Specifically, MRT-Red is the decreasing MRT where links in the GADAG are taken in the direction from a higher topologically ordered node to a lower one.

MRT-Blue: MRT-Blue is used to describe one of the two MRTs; it is used to described the associated forwarding topology and MT-ID. Specifically, MRT-Blue is the increasing MRT where links in the GADAG are taken in the direction from a lower topologically ordered node to a higher one.

MRT Island: From the computing router, the set of routers that support a particular MRT profile and are connected via MRT-eligible links.

Island Border Router (IBR): A router in the MRT Island that is connected to a router not in the MRT Island and both routers are in a common area or level.

Island Neighbor (IN): A router that is not in the MRT Island but is adjacent to an IBR and in the same area/level as the IBR.

4. MRT for Segment Routing Network

4.1. Overview

Segment Routing (SR) allows a flexible definition of end-to-end paths within IGP topologies by encoding paths as sequences of topological sub-paths, called "segments". These segments are advertised by the link-state routing protocols (IS-IS and OSPF). Prefix segments represent an ECMP-aware shortest-path to a prefix (or a node), as per the state of the IGP topology. Adjacency segments represent a hop over a specific adjacency between two nodes in the IGP.

MRT Fast Reroute requires that packets to be forwarded not only on the shortest-path tree, but also on two Maximally Redundant Trees (MRTs), referred to as the MRT-Blue and the MRT-Red. A router that experiences a local failure must also have predetermined which alternate to use. The MRT algorithm is defined in [RFC7811]. The

Default MRT Profile path calculation uses Lowpoint algorithm to calculate Maximally Redundant Trees.

To use MRT in Segment Routing network below mentioned capabilities needs to be incorporated in SR node.

1. SR nodes MUST support IGP extensions for MRT.

2. SR nodes MUST support MRT-RED and MRT-BLUE Algorithms.

3. SR nodes MUST advertise it’s MRT capability.

4. SR nodes SHOULD send and receive SID/Label for MRT topologies (MRT-RED and MRT-BLUE) for SR segment(s).

5. SR nodes MUST support computation of MRTs.

4.2. IGP extensions for MRT

These extensions are to support the distributed computation of Maximally Redundant Trees (MRT). These extensions indicate the MRT profile(s) each router supports. Different MRT profiles can be defined to support different uses and to allow transition of capabilities.

To support MRT for SR, a new SR MRT Profile is defined (as defined in section 5 of this document). IGP extensions for MRT[I-D.ietf-isis-mrt] / [I-D.ietf-ospf-mrt]MUST carry SR MRT profile.

4.3. SR Algorithm value for MRT-Blue and MRT-Red

Segment routing supports the use of different algorithms for computing paths to reach nodes and prefixes. To accomplish this, every Prefix-SID has a mandatory algorithm field. This Prefix-SID identifies an instruction to forward a packet along the path computed using the algorithm identified in the algorithm field.

[I-D.ietf-spring-segment-routing] defines two algorithms, Shortest Path and Strict Shortest Path.
[I-D.ietf-isis-segment-routing-extensions] and
[I-D.ietf-ospf-segment-routing-extensions] each assign the algorithm values of 0 and 1 to identify these two algorithms.

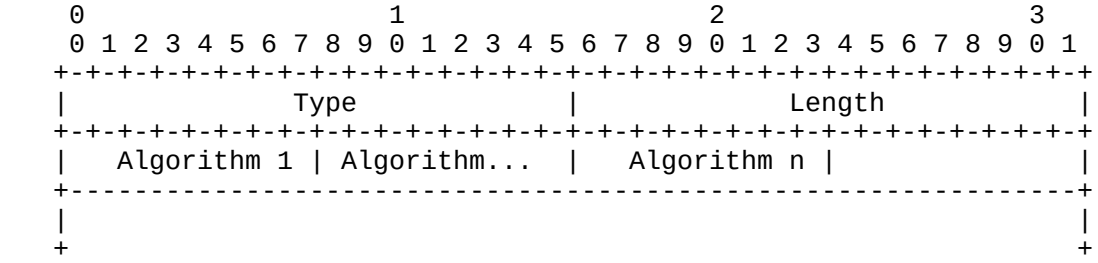
This document defines two additional algorithm values to support MRT-FRR using Segment Routing. The two algorithms correspond to the MRT-Red and MRT-Blue for the Default MRT Profile.

4.4. MRT capability advertisement for SR

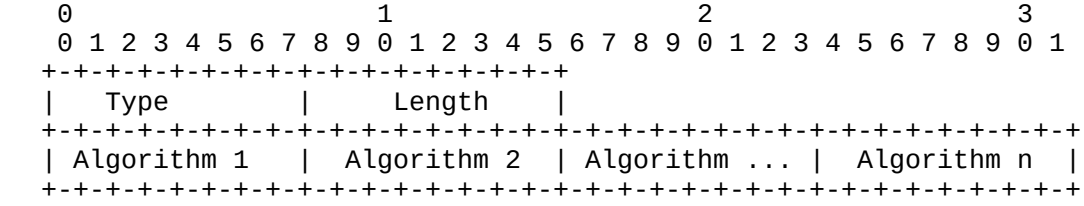
As packets routed on a hop-by-hop basis require that each router compute a shortest-path tree that is consistent, it is necessary for each router to compute the MRT-Blue next hops and MRT-Red next hops in a consistent fashion. For this each SR node needs to know which of the SR nodes in the SR domain supports MRT. This MUST be communicated using SR MRT Capability Advertisement.

Both OSPF [I-D.ietf-ospf-segment-routing-extensions] and ISIS [I-D.ietf-isis-segment-routing-extensions] supports segment routing capabilities advertisement. MRT algorithm capabilities need to be advertised in SR-Algorithm TLV for OSPF extension for segment routing and SR-Algorithm Sub-TLV for ISIS extension for segment routing.

SR-Algorithm TLV [I-D.ietf-ospf-segment-routing-extensions]



SR-Algorithm Sub-TLV[I-D.ietf-isis-segment-routing-extensions]



SR node is considered as MRT capable node if it advertises MRT capability in IGP extension of MRT as well as IGP extension of SR.

4.5. SR MRT SID/Label advertisement

In a link-state IGP domain, an SR-capable IGP node advertises segments for its attached prefixes and adjacencies. These segments are called IGP segments or IGP SIDs. They play a key role in Segment Routing as they enable the expression of any topological path throughout the IGP domain. Such a topological path is either expressed as a single IGP segment or a list of multiple IGP segments.

SR nodes supporting MRT MUST advertise two additional labels corresponding to MRT-BLUE and MRT-RED for each prefix segment.

These labels are carried as a part of prefix SID sub-tlv (OSPF Section 5 of [I-D.ietf-ospf-segment-routing-extensions], ISIS Section 2.1 of [I-D.ietf-isis-segment-routing-extensions]) with algorithm field set to algorithm value corresponding to the MRT forwarding topology as described in section Section 4.3.

4.6. MRT computation for SR

An SR node that advertise support for the Segment Routing MRT Profile MUST support MRT-RED and MRT-BLUE forwarding topology creation and support the computation of FRR paths as per the MRT algorithm described in [RFC7811].

5. MRT-FRR for destination-based and traffic-engineered SR

In addition to the Prefix-SIDs for Shortest Path algorithm, the IGP distributes Prefix-SIDs for MRT-Red and MRT-Blue. This allows each node to install transit forwarding entries for the MRT-Red and MRT-Blue paths for those prefixes. In normal operation, the traffic for a given destination will be forwarded based on the Shortest Path algorithm Prefix-SID for that destination. Upon detecting a link failure, a node can act as a point-of-local repair (PLR) by forwarding the traffic using either the MRT-Red or MRT-Blue Prefix-SID for the same destination. Following the appropriate MRT path, the traffic will the destination without crossing the failed link or the remote node attached to the failed link, if such a path exists.

The description above is independent of the segment routing forwarding plane. With the MRT-Red and MRT-Blue Segment Routing Algorithm values defined in this document, MRT-FRR can provide protection for traffic using either the MPLS or the IPv6 forwarding plane for segment routing. For clarity, we also describe the MRT-FRR mechanism when realized using the MPLS forwarding plane for segment routing.

In the MPLS-specific description, each node uses the index information contained in Prefix-SIDs and the SRGBs advertised by its neighbors to install transit forwarding entries for the Shortest Path, MRT-Red path, and MRT-Blue path for each destination prefix. As an example, the transit forwarding entry for a destination prefix on the MRT-Red path is a label swap operation where the both the incoming and outgoing labels correspond to the MRT-Red labels on the MRT-Red path.

In the absence of failures, traffic flows on transit forwarding entries corresponding to the Shortest Path algorithm where an incoming Shortest Path label is swapped to an outgoing Shortest Path label for a given destination. Upon the failure of a link, the PLR may change some forwarding entries to swap an incoming Shortest Path label to an outgoing MRT-Red or MRT-Blue label.

MRT Prefix Segments are applicable to both destination-based SR as well as traffic-engineered SR. In the case of destination-based SR, the Segment List consists of a single Prefix Segment with an MRT-Red or MRT-Blue algorithm value. In this case Prefix Segment identifies the complete MRT-Red or MRT-Blue path to the destination node or prefix. In the case of traffic-engineered SR, a Prefix Segment with an MRT algorithm value may form part of a larger Segment List. In this case, the MRT Prefix Segment identifies a portion of the entire end-to-end path in the SR domain. That portion of the path corresponds to the MRT-Red or MRT-Blue path for that prefix.

6. SR MRT Profile

The following set of options defines the SR MRT Profile. The SR MRT Profile is indicated by the MRT Profile ID.

MRT Algorithm: MRT Lowpoint algorithm defined in [RFC7811].

MRT-Red SR Algorithm ID: The MRT-Red SR Algorithm ID is indicated by the MRT-Red SR Algorithm ID.

MRT-Blue SR Algorithm ID: The MRT-Blue SR Algorithm ID is indicated by the MRT-BLue SR Algorithm ID.

GADAG Root Selection Policy: Among the routers in the MRT Island with the lowest numerical value advertised for GADAG Root Selection Priority, an implementation MUST pick the router with the highest Router ID to be the GADAG root. Note that a lower numerical value for GADAG Root Selection Priority indicates a higher preference for selection.

Forwarding Mechanisms: MRT SR Label Option 1A

Recalculation: Recalculation of MRTs SHOULD occur as described in Section 12.2 of [RFC7812] with SR label.

Area/Level Border Behavior: As described in Section 10 of [RFC7812], ABRs/LBRs SHOULD ensure that traffic leaving the area also exits the MRT-Red or MRT-Blue forwarding topology.

7. IANA Considerations

IANA is requested to allocate a value from the MRT Profile Identifier Registry for the Segment Routing MRT Profile with a value of TDB-1.

Value	Description	Reference
TBD-1	Segment Routing MRT Profile	This document

Currently, there is no IANA registry defined for SR Algorithm values carried in the Prefix-SID sub-TLV and the SR Algorithm sub-TLV for IS-IS or the Prefix-SID sub-TLV and SR Algorithm TLV for OSPF [I-D.ietf-isis-segment-routing-extensions] and [I-D.ietf-ospf-segment-routing-extensions] define the Segment Routing algorithms for values 0 and 1.

This document requests IANA to create a registry entitled "Segment Routing Algorithm Values". This should appear in the IANA registry under a new top-level entry entitled "IGP-Independent Segment Routing Parameters". The initial registry is shown below.

Value	SR Algorithm	References
0	Shortest Path	I-D.ietf-isis-segment-routing-extensions I-D.ietf-ospf-segment-routing-extensions
1	Strict Shortest Path	I-D.ietf-isis-segment-routing-extensions I-D.ietf-ospf-segment-routing-extensions
TBD-2	MRT-Red	This document
TBD-3	MRT-Blue	This document

Note to RFC Editor: this section may be removed on publication as an RFC.

8. Security Considerations

Security consideration of MRT and SR are applicable here. None of the additional security consideration are identified.

9. Acknowledgements

None

10. References

10.1. Normative References

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

[RFC7811] Enyedi, G., Csaszar, A., Atlas, A., Bowers, C., and A. Gopalan, "An Algorithm for Computing IP/LDP Fast Reroute Using Maximally Redundant Trees (MRT-FRR)", RFC 7811, DOI 10.17487/RFC7811, June 2016, <http://www.rfc-editor.org/info/rfc7811>.

[RFC7812] Atlas, A., Bowers, C., and G. Enyedi, "An Architecture for IP/LDP Fast Reroute Using Maximally Redundant Trees (MRT-FRR)", RFC 7812, DOI 10.17487/RFC7812, June 2016, <http://www.rfc-editor.org/info/rfc7812>.

10.2. Informative References

[I-D.ietf-isis-mrt]
Li, Z., Wu, N., <>, Q., Atlas, A., Bowers, C., and J. Tantsura, "Intermediate System to Intermediate System (IS-IS) Extensions for Maximally Redundant Trees (MRT)", draft-ietf-isis-mrt-02 (work in progress), May 2016.

[I-D.ietf-isis-segment-routing-extensions]
Previdi, S., Filsfils, C., Bashandy, A., Gredler, H., Litkowski, S., Decraene, B., and J. Tantsura, "IS-IS Extensions for Segment Routing", draft-ietf-isis-segment-routing-extensions-07 (work in progress), June 2016.

[I-D.ietf-ospf-mrt]
Atlas, A., Hegde, S., Bowers, C., Tantsura, J., and Z. Li, "OSPF Extensions to Support Maximally Redundant Trees", draft-ietf-ospf-mrt-02 (work in progress), May 2016.

[I-D.ietf-ospf-segment-routing-extensions]
Psenak, P., Previdi, S., Filsfils, C., Gredler, H., Shakir, R., Henderickx, W., and J. Tantsura, "OSPF Extensions for Segment Routing", draft-ietf-ospf-segment-routing-extensions-09 (work in progress), July 2016.

[I-D.ietf-spring-segment-routing]
Filsfils, C., Previdi, S., Decraene, B., Litkowski, S., and R. Shakir, "Segment Routing Architecture", draft-ietf-spring-segment-routing-09 (work in progress), July 2016.

Authors' Addresses

Anil Kumar S N
Huawei Technologies
Near EPIP Industrial Area, Kundalahalli Village
Whitefield, Bangalore, Karnataka 560066
India

Email: anil.ietf@gmail.com

Gaurav Agrawal
Huawei Technologies
Near EPIP Industrial Area, Kundalahalli Village
Whitefield, Bangalore, Karnataka 560066
India

Email: gaurav.agrawal@huawei.com

Vinod Kumar S
Huawei Technologies
Near EPIP Industrial Area, Kundalahalli Village
Whitefield, Bangalore, Karnataka 560066
India

Email: vinods.kumar@huawei.com

Chris Bowers
Juniper Networks
1194 N. Mathilda Ave.
Sunnyvale, CA 94089
USA

Email: cbowers@juniper.net

Network Working Group
Internet Draft
Intended status: Informational
Expires: May 2016

A. Bashandy, Ed.
C. Filsfils
Cisco Systems
P. Mohapatra
Sproute Networks

November 9, 2015

BGP Prefix Independent Convergence
draft-bashandy-rtgwg-bgp-pic-02.txt

Abstract

In the network comprising thousands of iBGP peers exchanging millions of routes, many routes are reachable via more than one path. Given the large scaling targets, it is desirable to restore traffic after failure in a time period that does not depend on the number of BGP prefixes. In this document we proposed an architecture by which traffic can be re-routed to ECMP or pre-calculated backup paths in a timeframe that does not depend on the number of BGP prefixes. The objective is achieved through organizing the forwarding chains in a hierarchical manner and sharing forwarding elements among the maximum possible number of routes. The proposed technique achieves prefix independent convergence while ensuring incremental deployment, complete transparency and automation, and zero management and provisioning effort. It is noteworthy to mention that the benefits of BGP-PIC are hinged on the existence of more than one path whether as ECMP or primary-backup.

This Internet-Draft will expire on May 9, 2016.

Table of Contents

1. Introduction

As a path vector protocol, BGP is inherently slow due to the serial nature of reachability propagation. BGP speakers exchange reachability information about prefixes[2][3] and, for labeled address families, namely AFI/SAFI 1/4, 2/4, 1/128, and 2/128, an edge router assigns local labels to prefixes and associates the local label with each advertised prefix such as L3VPN [8], 6PE [9], and Softwire [7] using BGP label unicast technique[4]. A BGP speaker then applies the path selection steps to choose the best path. In modern networks, it is not uncommon to have a prefix reachable via multiple edge routers. In addition to proprietary techniques, multiple techniques have been proposed to allow for more than one path for a given prefix [6][11][12], whether in the form of equal cost multipath or primary-backup. Another more common and widely deployed scenario is L3VPN with multi-homed VPN sites.

This document proposes a hierarchical and shared forwarding chain organization that allows traffic to be restored to pre-calculated alternative equal cost primary path or backup path in a time period that does not depend on the number of BGP prefixes. The technique relies on internal router behavior that is completely transparent to the operator and can be incrementally deployed and enabled with zero operator intervention.

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 RFC-2119 [1].

In this document, these words will appear with that interpretation only when in ALL CAPS. Lower case uses of these words are not to be interpreted as carrying RFC-2119 significance.

1.2. Terminology

This section defines the terms used in this document. For ease of use, we will use terms similar to those used by L3VPN [8]

- BGP prefix: It is a prefix P/m (of any AFI/SAFI) that a BGP speaker has a path for.
- IGP prefix: It is a prefix P/m (of any AFI/SAFI) that is learnt via an Interior Gateway Protocol, such as OSPF and ISIS, has a path for. The prefix may be learnt directly through the IGP or redistributed from other protocol(s)
- CE: It is an external router through which an egress PE can reach a prefix P/m.
- Ingress PE, "iPE": It is a BGP speaker that learns about a prefix through another IBGP peer and chooses that IBGP peer as the next-hop for the prefix.
- Path: It is the next-hop in a sequence of unique connected nodes starting from the current node and ending with the destination node or network identified by the prefix.
- Recursive path: It is a path consisting only of the IP address of the next-hop without the outgoing interface. Subsequent lookups are needed to determine the outgoing interface.
- Non-recursive path: It is a path consisting of the IP address of the next-hop and one outgoing interface
- Primary path: It is a recursive or non-recursive path that can be used all the time. A prefix can have more than one primary path
- Backup path: It is a recursive or non-recursive path that can be used only after some or all primary paths become unreachable
- Leaf: A leaf is container data structure for a prefix or local label. Alternatively, it is the data structure that contains prefix specific information.
- IP leaf: Is the leaf corresponding to an IPv4 or IPv6 prefix
- Label leaf. It is the leaf corresponding to a locally allocated label such as the VPN label on an egress PE [8].

- Pathlist: It is an array of paths used by one or more prefix to forward traffic to destination(s) covered by a IP prefix. Each path in the pathlist carries its "path-index" that identifies its position in the array of paths. A pathlist may contain a mix of primary and backup paths

- OutLabel-Array: Each labeled prefix is associated with an OutLabel-Array. The OutLabel-Array is a list of one or more outgoing labels and/or label actions where each label or label action has 1-to-1 correspondence to a path in the pathlist. It is possible that the number of entries in the OutLabel-array is different from the number of paths in the pathlist and the ith Outlabel-Array entry is associated with the path whose path-index is "i". Label actions are: push the label, pop the label, or swap the incoming label with the label in the Outlabel-Array entry. The prefix may be an IGP or BGP prefix

- Adjacency: It is the layer 2 encapsulation leading to the layer 3 directly connected next-hop

- Dependency: An object X is said to be a dependent or Child of object Y if Object Y cannot be deleted unless object X is no longer a dependent/child of object Y

- Route: It is a prefix with one or more paths associated with it. Hence the minimum set of objects needed to construct a route is a leaf and a pathlist.

2. Constructing the Shared Hierarchical Forwarding Chain

2.1. Databases

The Forwarding Information Base (FIB) on a router maintains 3 basic databases

- Pathlist-DB: A pathlist is uniquely identified by the list of paths. The Pathlist DB contains the set of all shared pathlists

- Leaf-DB: A leaf is uniquely identified by the prefix or the label

- Adjacency-DB: An adjacency is uniquely identified by the outgoing layer 3 interface and the IP address of the next-hop directly connected to the layer 3 interface. Adjacency DB contains the list of all adjacencies

2.2. Constructing the forwarding chain from a downloaded route

1. A prefix with a list of paths is downloaded to FIB from BGP. For labeled prefixes, an OutLabel-Array and possibly a local label (e.g. for a VPN [8] prefix on an egress PE) are also downloaded

2. If the prefix does not exist, construct a new IP leaf from the downloaded prefix. If a local label is allocated, construct a label leaf from the local label

3. Construct an OutLabel-Array and attach the Outlabel array to the IP and label leaf

4. The list of paths attached to the route is looked up in the pathlist-DB

5. If a pathlist PL is found

    a. Retrieve the pathlist

6. Else

    a. Construct a new pathlist

    b. Insert the new pathlist in the pathlist-DB

    c. Resolve the paths of the pathlist as follows

    d. Recursive path:

        i. Lookup the next-hop in the leaf-DB

        ii. If a leaf with at least one reachable path is found, add the path to the dependency list of the leaf

        iii. Otherwise the path remains unresolved and cannot be used for forwarding

    e. Non-recursive path

        i. Lookup the next-hop and outgoing interface in the adjacency-DB

        ii. If an adjacency is found, add the path to the dependency list of adjacency

        iii. Otherwise, create a new adjacency and add the path to its dependency list

7. Attach the leaf(s) as (a) dependent(s) of the pathlist

As a result of the above steps, a forwarding chain starting with a leaf and ending with one or more adjacency is constructed. It is noteworthy to mention that the forwarding chain is constructed without any operator intervention at all.

2.3. Examples

This section outlines three examples that we will use for illustration for the rest of the document. The first two examples use a standard multihomed VPN [8] prefix in a BGP-free core running LDP [5] or segment routing on MPLS [14]. The third example uses inter-AS option C [8] with 2 domains running segment routing [14] or LDP [5] in the core

The topology for the first two examples is depicted in Figure 1.

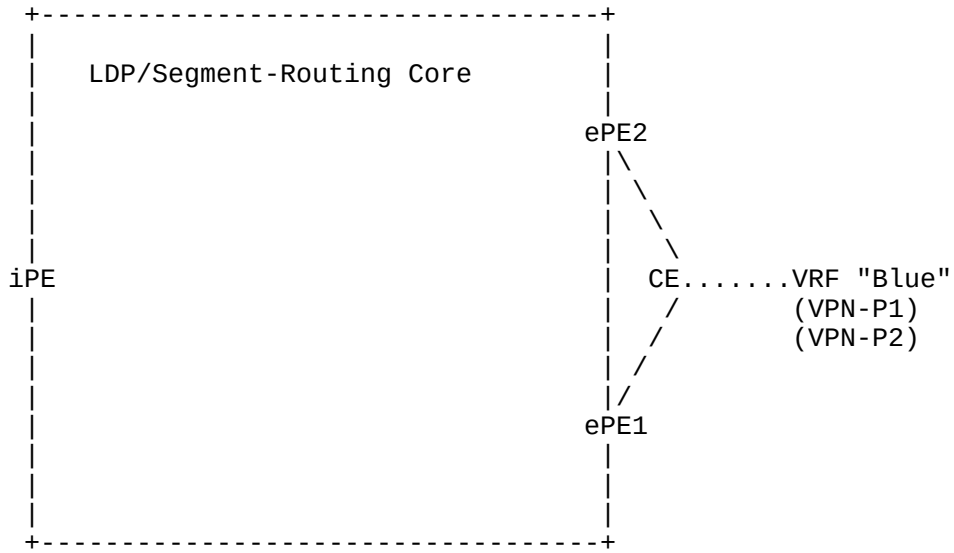


Figure 1 VPN prefix reachable via multiple PEs

The first example is an illustration of ECMP while the second example is an illustration of primary-backup paths. The third example illustrate how to handle limited hardware capability.

2.3.1. Example 1: Forwarding Chain for iBGP ECMP

Consider the case of the ingress PE (iPE) in the multi-homed VPN prefixes depicted in Figure 1. Suppose the iPE receives route advertisements for the VPN prefixes VPN-P1 and VPN-P2 from two egress PEs, ePE1 and ePE2 with next-hop BGP-NH1 and BGP-NH2, respectively. Assume that ePE1 advertise the VPN labels VPN-L11 and VPN-L12 while ePE2 advertise the VPN labels VPN-L21 and VPN-L22 for

VPN-P1 and VPN-P2, respectively. Suppose that BGP-NH1 and BGP-NH2 are resolved via the IGP prefixes IGP-P1 and IGP-P2, which also happen to have 2 ECMP paths with IGP-NH1 and IGP-NH2 reachable via the interfaces I1 and I2. Suppose that local labels (whether LDP[5] or segment routing [14]) on the downstream LSRs for IGP-P1 and IGP-P2 are assign the LDP labels LDP-L1 and LDP-L2 to the prefixes IGP-P1 and IGP-P2. The forwarding chain on the ingress PE "iPE" for the VPN prefixes is depicted in Figure 2.

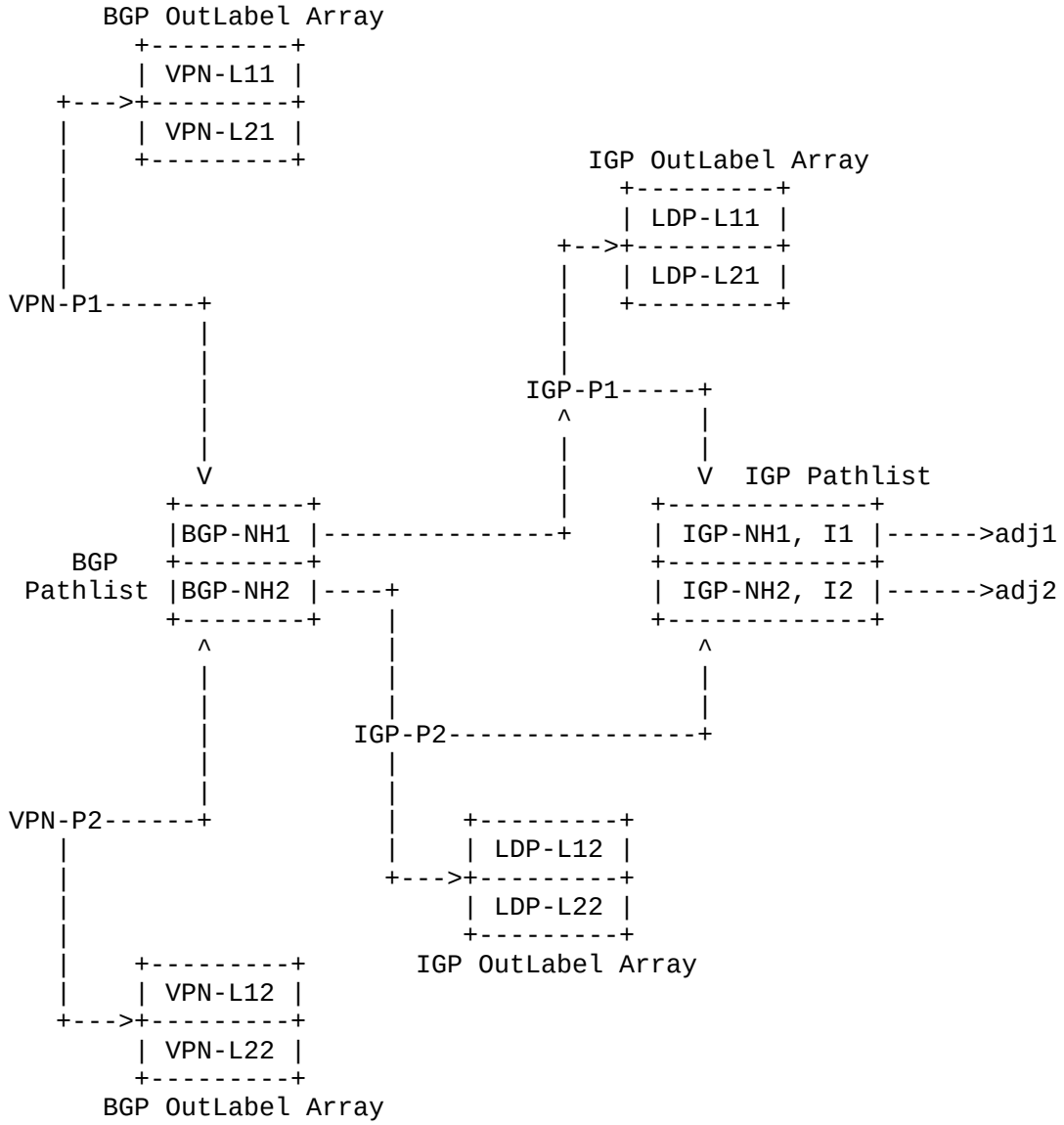


Figure 2 Forwarding Chain for VPN Prefixes with iBGP ECMP

The structure depicted in Figure 2 illustrates the two important properties discussed in this memo: sharing and hierarchy. We can see that the both the BGP and IGP pathlists are shared among multiple BGP and IGP prefixes, respectively. At the same time, the forwarding chain objects depend on each other in a child-parent relation instead of being collapsed into a single level.

2.3.2. Example 2: Primary Backup Paths

Consider the egress PE ePE1 in the case of the multi-homed VPN prefixes in the BGP-free LDP core depicted in Figure 1. Suppose ePE1 determines that the primary path is the external path but the backup path is the iBGP path to the other PE ePE2 with next-hop BGP-NH2. ePE2 constructs the forwarding chain depicted in Figure 1. We are only showing a single VPN prefix for simplicity. But all prefixes that are multihomed to ePE1 and ePE2 share the BGP pathlist

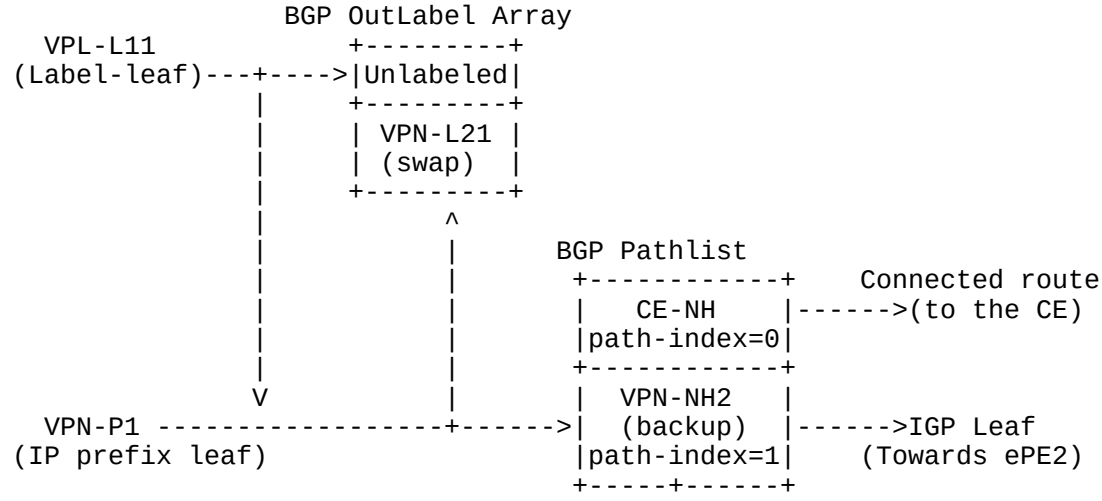


Figure 3 : VPN Prefix Forwarding Chain with eiBGP paths on egress PE

The example depicted in Figure 3 differs from the example in Figure 2 in two main aspects. First as long as the primary path towards the CE (external path) is useable, it will be the only path used for forwarding while the OutLabel-Array contains both the unlabeled label (primary path) and the VPN label (backup path) advertised by the backup path ePE2. The second aspect is presence of the label leaf corresponding to the VPN prefix. This label leaf is used to match VPN traffic arriving from the core. Note that the label leaf shares the OutLabel-Array and the pathlist with the IP prefix.

2.3.3. Example 3: Platforms with Limited Levels of Hierarchy

This example uses a case of inter-AS option C [8] where there are 3 levels of hierarchy. Figure 4 illustrates the sample topology. To force 3 levels of hierarchy, the ASBRs on the ingress domain (domain 1) advertise the core routers of the egress domain (domain 2) to the ingress PE (iPE) via BGP-LU [4] instead of redistributing then into

the IGP of domain 1. The end result is that the ingress PE (iPE) has 2 levels of recursion for the VPN prefix VPN-P1 and VPN2-P2.

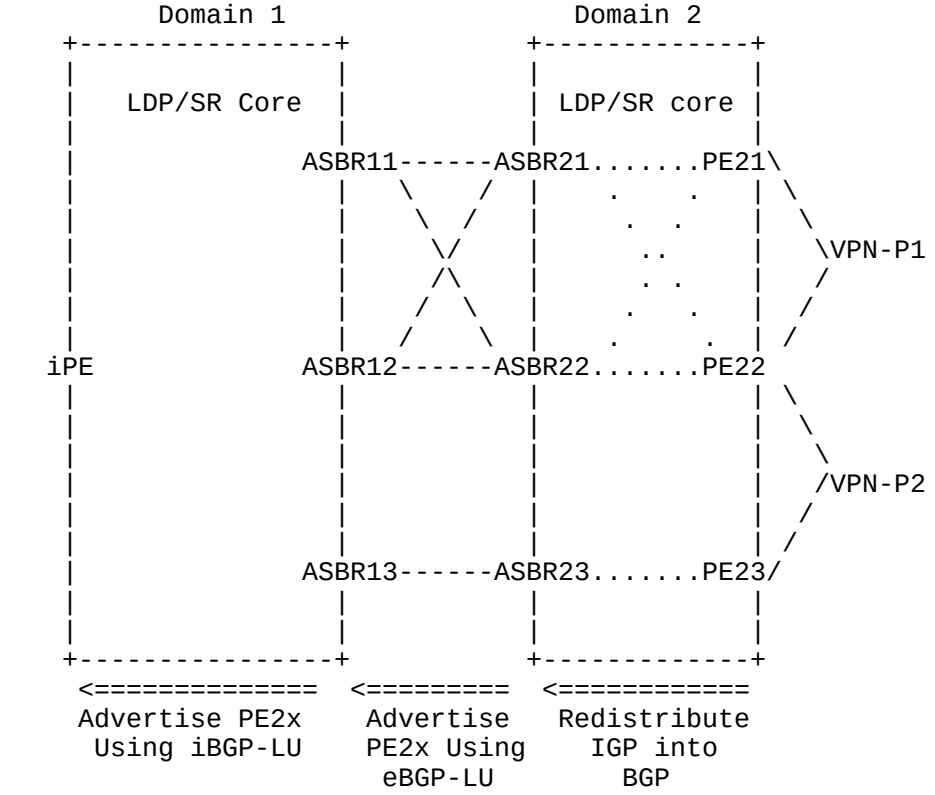


Figure 4 Sample 3-level hierarchy topology

We will make the following assumptions about connectivity

- In "domain 2", both ASBR21 and ASBR22 can reach both PE21 and PE22 using the same distance
- In "domain 2", only ASBR23 can reach PE23
- In "domain 1", iPE (the ingress PE) can reach ASBR1, ASBR12, and ASBR13 via IGP using the same distance

We will make the following assumptions about the labels

- The VPN labels advertised by PE21 and PE22 for prefix VPN-P1 are VPN-PE21(P1) and VPN-PE22(P1), respectively

- The VPN labels advertised byPE22 and PE23 for prefix VPN-P2 are VPN-PE22(P2) and VPN-PE23(P2), respectively

- The labels for advertised to iPE by ASBR11 using BGP-LU [4] for the egress PEs PE21 and PE22 are LASBR11(PE21) and LASBR11(PE22), respectively.

- The labels for advertised by ASBR12 to iPE using BGP-LU [4] for the egress PEs PE21 and PE22 are LASBR12(PE21) and LASBR12(PE22), respectively

- The label for advertised by ASBR11 to iPE using BGP-LU [4] for the egress PE PE23 is LASBR13(PE23)

- The local labels of the next hops from the ingress PE iPE towards ASBR11, ASBR12, and ASBR13 in the core of domain 1 are L11, L12, and L13, respectively.

The diagram in Figure 5 illustrates the forwarding chain assuming that the forwarding hardware in iPE supports 3 levels of hierarchy. The leaves corresponding to the ABSRs on domain 1 (ASBR11, ASBR12, and ASBR13) are at the bottom of the hierarchy. There are few important points

- Because the hardware supports the required depth of hierarchy, the sizes of a pathlist equal the size of the label array associated with the leaves using this pathlist

- The index inside the pathlist entry indicates the label that will be picked from the Outlabel-array if that path is chosen by the forwarding engine hashing function.

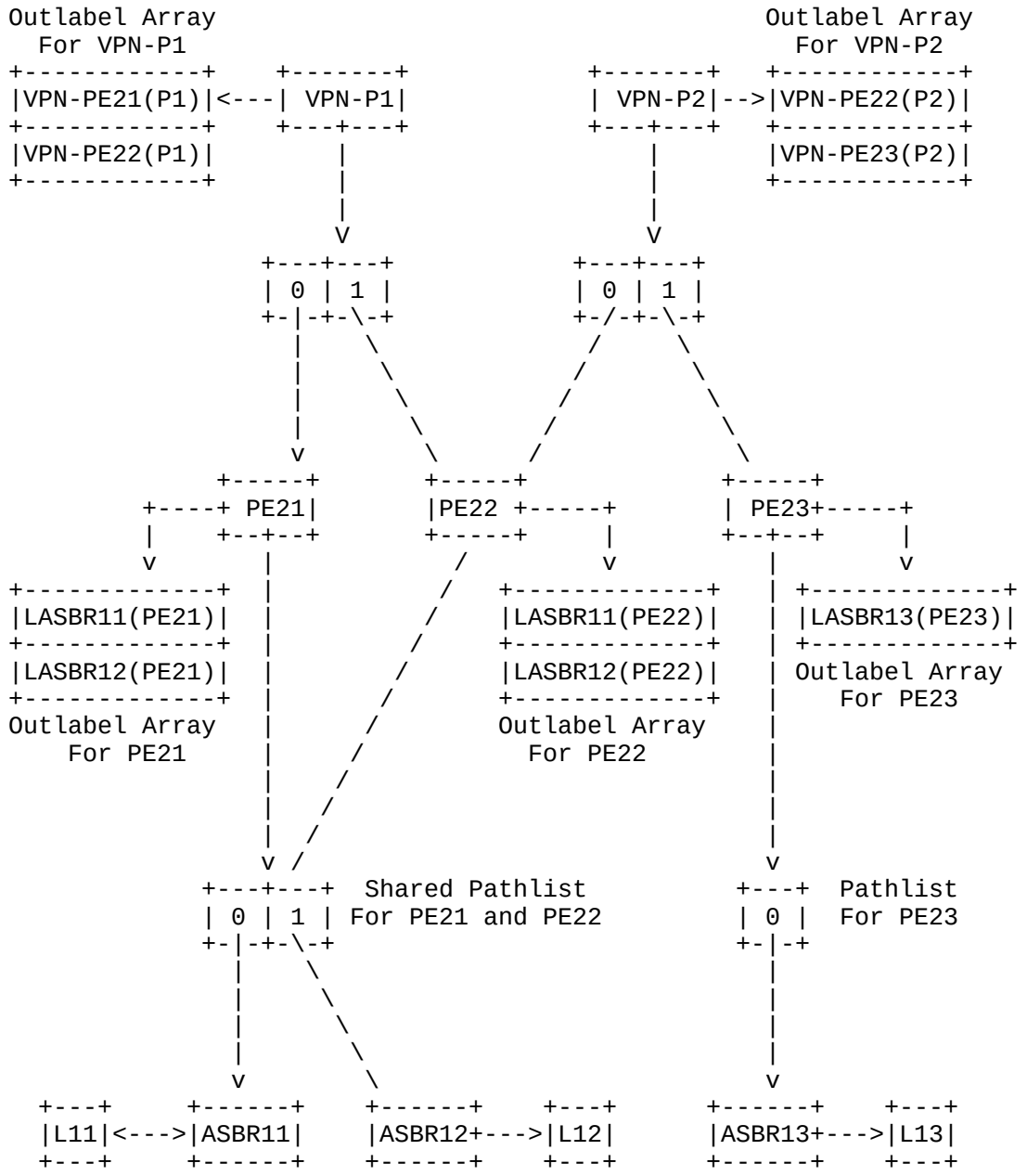


Figure 5 : Forwarding Chain for hardware supporting 3 Levels

Now suppose the hardware on iPE (the ingress PE) supports 2 levels of hierarchy only. In that case, the 3-levels forwarding chain in Figure 5 needs to be "flattended" into 2 levels only.

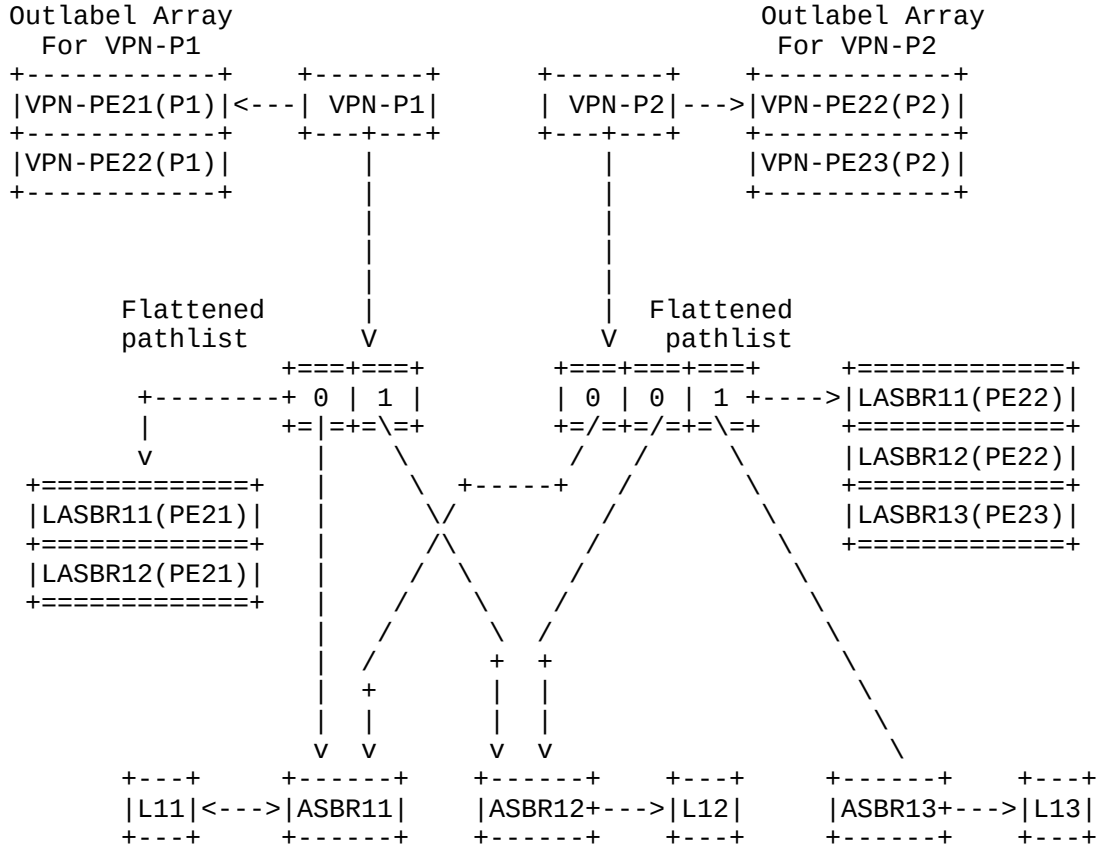


Figure 6 : Flattening 3 levels to 2 levels of Hierarchy on iPE

Figure 6 represents one way to "flatten" a 3 levels hierarchy into two levels. There are few important points.

o The flattened pathlists have label arrays associated with them. The size of the label array associated with the flattened pathlist equals the size of the pathlist. Hence it is possible that an implementation includes these label arrays in the flattened pathlist itself

- Because of "flattening", the size of a flattened pathlist may not be equal to the size of the label arrays of leaves using the flattened pathlist.

- The indices inside a flattened pathlist still indicate the label index in the Outlabel-Arrays of the leaves using that pathlist. Because the size of the flattened pathlist may be different from the size of the label arrays of the leaves, the indices may be repeated

- Let’s take a look at the flattened pathlist used by the prefix "VPN-P2", The pathlist associated with the prefix "VPN-P2" has three entries.

    - The first and second entry have index "0". This is because both entries correspond to PE22. Hence when hashing performed by the forwarding engine results in using first or the second entry in the pathlist, the forwarding engine will pick the correct VPN label "VPN-PE22(P2)", which is the label advertised by PE22 for the prefix "VPN-P2"

    - The third entry has the index "1". This is because the third entry corresponds to PE23. Hence when the hashing is performed by the forwarding engine results in using the third entry in the flattened pathlist, the forwarding engine will pick the correct VPN label "VPN-PE22(P2)", which is the label advertised by "PE23" for the prefix "VPN-P2"

3. Forwarding Behavior

When a packet arrives at a router, it matches a leaf. A labeled packet matches a label leaf while an IP packet matches an IP prefix leaf. The forwarding engines walks the forwarding chain starting from the leaf until the walk terminates on an adjacency. Thus when a packet arrives, the chain is walked as follows:

1. Lookup the leaf based on the destination address or the label at the top of the packet

2. Retrieve the parent pathlist of the leaf

3. Pick the outgoing path from the list of resolved paths in the pathlist. The method by which the outgoing path is picked is beyond the scope of this document (i.e. flow-preserving hash exploiting entropy within the MPLS stack and IP header). Let the "path-index" of the outgoing path be "i".

4. If the prefix is labeled, use the "path-index" "i" to retrieve the ith label "Li" stored the ith entry in the OutLabel-Array and apply the label action of the label on the packet (e.g. for VPN label on the ingress PE, the label action is "push").

5. Move to the parent of the chosen path "i"

6. If the chosen path "i" is recursive, move to its parent prefix and go to step 2

7. If the chosen path "i" is non-recursive move to its parent adjacency

8. Encapsulate the packet in the L2 string specified by the adjacency and send the packet out.

Let’s applying the above forwarding steps to the example described in Figure 1 Section 2.3.1. Suppose a packet arrives at ingress PE iPE from an external neighbor. Assume the packet matches the VPN prefix VPN-P1. While walking the forwarding chain, the forwarding engine applies a hashing algorithm to choose the path and the hashing at the BGP level yields path 0 while the hashing at the IGP level yields path 1. In that case, the packet will be sent out of interface I1 with the label stack "LDP-L12,VPN-L21".

Now let’s try and apply the above steps to the flattened forwarding chain illustrated in Figure 6.

- Suppose a packet arrives at "iPE" and matches the VPN prefix "VPN-P2"

- The forwarding engine walks to the parent of the "VPN_P2", whiuch is the flattened pathlist and applies a hashing algorithm to pick a path

- Suppose the hashing by the forwarding engine picks the second entry in the flattened pathlist associated with the leaf "VPN-P2".

- Because the second entry has the index "0", the label "VPN-PE22(P2)" is pushed on the packet

- At the same time, the forwarding engine picks the second label from the Outlabel-Array associated with the flattened pathlist. Hence the next label that is pushed is "LASBR12(PE22)"

- The forwarding engine now moves to the parent of the flattened pathlist corresponding tgo the second entry. The parent is the IGP label leaf corresponding to "ASBR12"

- So the packet is forwarded towards the ASBR "ASBR12" and the SR/LDP label at the top will be "L12"

The packet is arriving at iPE reaches its destination as follows

- iPE sends the packet along the shortest path towards ASBR12 with the following label stack starting from the top: {L12, LASBR12(PE22), VPN-PE22(P2)}.
- The penultimate hop of ASBR12 pops the top label "L12". Hence the packet arrives at ASBR12 with the label stack {LASBR12(PE22), VPN-PE22(P2)} where "LASBR12(PE22)" is the top label.
- ASBR12 swaps "LASBR12(PE22)" with the label "LASBR22(PE22)", which is the label advertised by ASBR22 for the PE22 (the egress PE).
- ASBR22 receives the packet with "LASBR22(PE22)" at the top.
- Hence ASBR22 swaps "LASBR22(PE22)" with the LDP/SR label of PE22, pushes the label of the next-hop towards PE22 in domain 2, and sends the packet along the shortest path towards PE22.
- The penultimate hop of PE22 pops the top label. Hence PE22 receives the packet with the top label VPN-PE22(P2) at the top
- PE22 pops "VPN-PE22(P2)" and sends the packet as a pure IP packet towards the destination VPN-PE22.

4. Forwarding Chain Adjustment at a Failure

The hierarchical and shared structure of the forwarding chain explained in Section 2 allows modifying a small number of forwarding chain objects to re-route traffic to a pre-calculated equal-cost or backup path without the need to modify the possibly very large number of BGP prefixes. In this section, we go over various core and edge failure scenarios to illustrate how FIB manager can utilize the forwarding chain structure to achieve prefix independent convergence.

4.1. BGP-PIC core

This section describes the adjustments to the forwarding chain when a core link or node fails but the BGP next-hop remains reachable.

There are two case: remote link failure and attached link failure. Node failures are treated as link failures.

When a remote link or node fails, IGP on the ingress PE receives advertisement indicating a topology change so IGP re-converges to

either find a new next-hop and outgoing interface or remove the path completely from the IGP prefix used to resolve BGP next-hops. IGP and/or LDP download the modified IGP leaves with modified outgoing labels for labeled core. FIB manager modifies the existing IGP leaf by executing the steps outlined in Section 2.2.

When a local link fails, FIB manager detects the failure almost immediately. The FIB manager marks the impacted path(s) as unuseable so that only useable paths are used to forward packets. Note that in this particular case there is actually no need even to backwalk to IGP leaves to adjust the OutLabel-Arrays because FIB can rely on the path-index stored in the useable paths in the loadinfo to pick the right label.

It is noteworthy to mention that because FIB manager modifies the forwarding chain starting from the IGP leaves only, BGP pathlists and leaves are not modified. Hence traffic restoration occurs within the time frame of IGP convergence, and, for local link failure, within the timeframe of local detection. Thus it is possible to achieve sub-50 msec convergence as described in [10] for local link failure

Let’s apply the procedure to the forwarding chain depicted in Figure 2 Section 2.3.1. Suppose a remote link failure occurs and impacts the first ECMP IGP path to the remote BGP nhop. Upon IGP convergence, the IGP pathlist of the BGP nhop is updated to reflect the new topology (one path instead of two). As soon as the IGP convergence is effective for the BGP nhop entry, the new forwarding state is immediately available to all dependent BGP prefixes. The same behavior would occur if the failure was local such as an interface going down. As soon as the IGP convergence is complete for the BGP nhop IGP route, all its BGP depending routes benefit from the new path. In fact, upon local failure, if LFA protection is enabled for the IGP route to the BGP nhop and a backup path was pre-computed and installed in the pathlist, upon the local interface failure, the LFA backup path is immediately activated (sub-50msec) and thus protection benefits all the depending BGP traffic through the hierarchical forwarding dependency between the routes.

4.2. BGP-PIC edge

This section describes the adjustments to the forwarding chains as a result of edge node or edge link failure

4.2.1. Adjusting forwarding Chain in egress node failure

When an edge node fails, IGP on neighboring core nodes send route updates indicating that the edge node is no longer reachable. IGP running on the iBGP peers instructs FIB to remove the IP and label leaves corresponding to the failed edge node from FIB. So FIB manager performs the following steps:

- FIB manager deletes the IGP leaf corresponding to the failed edge node
- FIB manager backwalks to all dependent BGP pathlists and marks that path using the deleted IGP leaf as unresolved
- Note that there is no need to modify BGP leaves because each path in the pathlist carries its path index and hence the correct outgoing label will be picked. So for example the forwarding chain depicted in Figure 2, if the 1st path becomes unresolved, then the forwarding engine will only use the second path path for forwarding. Yet the pathindex of that single resolved path will still be 1 and hence the label VPN-L21 or VPN-L22 will be pushed

4.2.2. Adjusting Forwarding Chain on PE-CE link Failure

Suppose the link between an edge router and its external peer fails. There are two scenarios (1) the edge node attached to the failed link performs next-hop self and (2) the edge node attached to the failure advertises the IP address of the failed link as the next-hop attribute to its iBGP peers.

In the first case, the rest of iBGP peers will remain unaware of the link failure and will continue to forward traffic to the edge node until the edge node attached to the failed link withdraws the BGP prefixes. If the destination prefixes are multi-homed to another iBGP peer, say ePE2, then FIB manager on the edge router detecting the link failure performs the following tasks

- FIB manager backwalks to the BGP pathlists marks the path through the failed link to the external peer as unresolved
- Hence traffic will be forwarded used the backup path towards ePE2
- For labeled traffic
    - The Outlabel-Array attached to the BGP leaves already contains an entry corresponding to the path towards ePE2.
    - The label entry in OutLabel-Arrays corresponding to the internal path to ePE2 has swap action and the label advertised by ePE2

o For an arriving label packet (e.g. VPN), the top label is swapped with the label advertised by ePE2

- For unlabeled traffic, packets is simply redirected towards ePE2. To avoid loops, ePE2 MUST treat any core facing path as a backup path, otherwise ePE2 may redirect traffic arriving from the core back to ePE1 causing a loop.

In the second case where the edge router uses the IP address of the failed link as the BGP next-hop, the edge router will still perform the previous steps. But, unlike the case of next-hop self, IGP on failed edge node informs the rest of the iBGP peers that IP address of the failed link is no longer reachable. Hence the FIB manager on iBGP peers will delete the IGP leaf corresponding to the IP prefix of the failed link. The behavior of the iBGP peers will be identical to the case of edge node failure outlined in Section 4.2.1.

It is noteworthy to mention that because the edge link failure is local to the edge router, sub-50 msec convergence can be achieved as described in [10].

Let’s try to apply the case of next-hop self to the forwarding chain depicted in Figure 3. After failure of the link between ePE1 and CE, the forwarding engine will route traffic arriving from the core towards VPN-NH2 with path-index=1. A packet arriving from the core will contain the label VPN-L11 at top. The label VPN-L11 is swaped with the label VPN-L21 and the packet is forwarded towards ePE2

4.3. Handling Failures for Flattended Forwarding Chains

As explained in the Example in Section 2.3.3, if the number of hierarchy levels of a platform cannot support the number of hierarchy levels of a recursive dependency, the instantiated forwarding chain is constructed by flattening two or more levels. Hence a 3 levels chain in Figure 5 is flattened into the 2 levels chain in Figure 6.

While reducing the benefits of BGP-PIC, flattening one hierarchy into a shallower hierarchy does not always result in a complete loss of the benefits of the BGP-PIC. To illustrate this fact suppose ASBR12 is no longer reachable. If the platform supports the full hierarchy depth, the forwarding chain is depicted in Figure 5 and hence the FIB manager needs to backwalk one level to the pathlist shared by "PE21" and "PE222" and adjust it. If the platform supports 2 levels of hierarchy, then a useable forwarding chain is the one depicted in Figure 6. In that case, if ASBR12 is no longer reachable, the FIB manager has to backwalk to the two flattened pathlists and update both of them.

Hence if the platform supports the "unflattened" forwarding chain, then a single pathlist needs to be updated while if the platform supports a shallower forwarding chain, then two pathlists need to be updated. In the latter case, convergence is still independent of the number of leaves due to the fact that the flattened pathlists continue to be shared among possibly a large number of leaves

5. Properties

5.1 Coverage

All the possible failures, except CE node failure, are covered, whether they impact a local or remote IGP path or a local or remote BGP nhop as described in Section 4. This section provides details for each failure and now the hierarchical and shared FIB structure proposed in this document allows recovery that does not depend on number of BGP prefixes

5.1.1 A remote failure on the path to a BGP nhop

Upon IGP convergence, the IGP leaf for the BGP nhop is updated upon IGP convergence and all the BGP depending routes leverage the new IGP forwarding state immediately.

This BGP resiliency property only depends on IGP convergence and is independent of the number of BGP prefixes impacted.

5.1.2 A local failure on the path to a BGP nhop

Upon LFA protection, the IGP leaf for the BGP nhop is updated to use the precomputed LFA backup path and all the BGP depending routes leverage this LFA protection.

This BGP resiliency property only depends on LFA protection and is independent of the number of BGP prefixes impacted.

5.1.3 A remote iBGP nhop fails

Upon IGP convergence, the IGP leaf for the BGP nhop is deleted and all the depending BGP Path-Lists are updated to either use the remaining ECMP BGP best-paths or if none remains available to activate precomputed backups.

This BGP resiliency property only depends on IGP convergence and is independent of the number of BGP prefixes impacted.

5.1.4 A local eBGP nhop fails

Upon local link failure detection, the adjacency to the BGP nhop is deleted and all the depending BGP Path-Lists are updated to either use the remaining ECMP BGP best-paths or if none remains available to activate precomputed backups.

This BGP resiliency property only depends on local link failure detection and is independent of the number of BGP prefixes impacted.

5.2 Performance

When the failure is local (a local IGP nhop failure or a local eBGP nhop failure), a pre-computed and pre-installed backup is activated by a local-protection mechanism that does not depend on the number of BGP destinations impacted by the failure. Sub-50msec is thus possible even if millions of BGP routes are impacted.

When the failure is remote (a remote IGP failure not impacting the BGP nhop or a remote BGP nhop failure), an alternate path is activated upon IGP convergence. All the impacted BGP destinations benefit from a working alternate path as soon as the IGP convergence occurs for their impacted BGP nhop even if millions of BGP routes are impacted.

5.2.1 Perspective

The following table puts the BGP PIC benefits in perspective assuming

- 1M impacted BGP prefixes
- IGP convergence ~ 500 msec
- local protection ~ 50msec
- FIB Update per BGP destination ~ 100usec conservative,

  ~ 10usec optimistic
- BGP Convergence per BGP destination ~ 200usec conservative,

  ~ 100usec optimistic

	Without PIC	With PIC
Local IGP Failure	10 to 100sec	50msec
Local BGP Failure	100 to 200sec	50msec

Remote IGP Failure	10 to 100sec	500msec
Local BGP Failure	100 to 200sec	500msec

Upon local IGP nhop failure or remote IGP nhop failure, the existing primary BGP nhop is intact and usable hence the resiliency only depends on the ability of the FIB mechanism to reflect the new path to the BGP nhop to the depending BGP destinations. Without BGP PIC, a conservative back-of-the-envelope estimation for this FIB update is 100usec per BGP destination. An optimistic estimation is 10usec per entry.

Upon local BGP nhop failure or remote BGP nhop failure, without the BGP PIC mechanism, a new BGP Best-Path needs to be recomputed and new updates need to be sent to peers. This depends on BGP processing time that will be shared between best-path computation, RIB update and peer update. A conservative back-of-the-envelope estimation for this is 200usec per BGP destination. An optimistic estimation is 100usec per entry.

5.3 Automated

The BGP PIC solution does not require any operator involvement. The process is entirely automated as part of the FIB implementation.

The salient points enabling this automation are:

- Extension of the BGP Best Path to compute more than one primary ([11]and [12]) or backup BGP nhop ([6] and [13]).
- Sharing of BGP Path-list across BGP destinations with same primary and backup BGP nhop
- Hierarchical indirection and dependency between BGP Path-List and IGP-Path-List

5.4 Incremental Deployment

As soon as one router supports BGP PIC solution, it benefits from all its benefits without any requirement for other routers to support BGP PIC.

6. Dependency

This section describes the required functionality in the forwarding and control planes to support BGP-PIC described in this document

6.1 Hierarchical Hardware FIB

BGP PIC requires a hierarchical hardware FIB support: for each BGP forwarded packet, a BGP leaf is looked up, then a BGP Pathlist is consulted, then an IGP Pathlist, then an Adjacency.

An alternative method consists in "flattening" the dependencies when programming the BGP destinations into HW FIB resulting in potentially eliminating both the BGP Path-List and IGP Path-List consultation. Such an approach decreases the number of memory lookup’s per forwarding operation at the expense of HW FIB memory increase (flattening means less sharing hence duplication), loss of ECMP properties (flattening means less pathlist entropy) and loss of BGP PIC properties.

6.2 Availability of more than one primary or secondary BGP next-hops

When the primary BGP next-hop fails, BGP PIC depends on the availability of a pre-computed and pre-installed secondary BGP next-hop in the BGP Pathlist.

The existence of a secondary next-hop is clear for the following reason: a service caring for network availability will require two disjoint network connections hence two BGP nhops.

The BGP distribution of the secondary next-hop is available thanks to the following BGP mechanisms: Add-Path [11], BGP Best-External [6], diverse path [12], and the frequent use in VPN deployments of different VPN RD’s per PE. It is noteworthy to mention that the availability of another BGP path does not mean that all failure scenarios can be covered by simply forwarding traffic to the available secondary path. The discussion of how to cover various failure scenarios is beyond the scope of this document

6.3 Pre-Computation of a secondary BGP nhop

[13] describes how a secondary BGP next-hop can be precomputed on a per BGP destination basis.

7. Security Considerations

No additional security risk is introduced by using the mechanisms proposed in this document

8. IANA Considerations

No requirements for IANA

9. Conclusions

This document proposes a hierarchical and shared forwarding chain structure that allows achieving prefix independent convergence, and in the case of locally detected failures, sub-50 msec convergence. A router can construct the forwarding chains in a completely transparent manner with zero operator intervention. It supports incremental deployment.

10. References

10.1. Normative References

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

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

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

[4] Y. Rekhter and E. Rosen, " Carrying Label Information in BGP-4", RFC 3107, May 2001

[5] Andersson, L., Minei, I., and B. Thomas, "LDP Specification", RFC 5036, October 2007

10.2. Informative References

[6] Marques,P., Fernando, R., Chen, E, Mohapatra, P., Gredler, H., "Advertisement of the best external route in BGP", draft-ietf-idr-best-external-05.txt, January 2012.

[7] Wu, J., Cui, Y., Metz, C., and E. Rosen, "Softwire Mesh Framework", RFC 5565, June 2009.

[8] Rosen, E. and Y. Rekhter, "BGP/MPLS IP Virtual Private Networks (VPNs)", RFC 4364, February 2006.

[9] De Clercq, J. , Ooms, D., Prevost, S., Le Faucheur, F., "Connecting IPv6 Islands over IPv4 MPLS Using IPv6 Provider Edge Routers (6PE)", RFC 4798, February 2007

[10] O. Bonaventure, C. Filsfils, and P. Francois. "Achieving sub-50 milliseconds recovery upon bgp peering link failures, " IEEE/ACM Transactions on Networking, 15(5):1123-1135, 2007

[11] D. Walton, E. Chen, A. Retana, J. Scudder, "Advertisement of Multiple Paths in BGP", draft-ietf-idr-add-paths-10.txt, October 2014

[12] R. Raszuk, R. Fernando, K. Patel, D. McPherson, K. Kumaki, "Distribution of diverse BGP paths", RFC 6774.txt, November 2012

[13] P. Mohapatra, R. Fernando, C. Filsfils, and R. Raszuk, "Fast Connectivity Restoration Using BGP Add-path", draft-pmohapat-idr-fast-conn-restore-03, Jan 2013

[14] C. Filsfils, S. Previdi, A. Bashandy, B. Decraene, S. Litkowski, M. Horneffer, R. Shakir, J. Tansura, E. Crabbe "Segment Routing with MPLS data plane", draft-ietf-spring-segment-routing-mpls-02 (work in progress), October 2015

11. Acknowledgments

Special thanks to Neeraj Malhotra, Yuri Tsier for the valuable help

Special thanks to Bruno Decraene for the valuable comments

This document was prepared using 2-Word-v2.0.template.dot.

Authors' Addresses

Ahmed Bashandy
Cisco Systems
170 West Tasman Dr, San Jose, CA 95134, USA
Email: bashandy@cisco.com

Clarence Filsfils
Cisco Systems
Brussels, Belgium
Email: cfilsfil@cisco.com

Prodosh Mohapatra
Sproute Networks
Email: mpradosh@yahoo.com

Network Working Group X. Chen
Internet-Draft Z. Li
Intended status: Standards Track Huawei Technologies
Expires: June 18, 2016 December 16, 2015

Yang Data Model for Resource Management
draft-chen-rtgwg-resource-management-yang-01

Abstract

This document describes a YANG data model for resource management. The resource includes mpls label etc. The data model defines the configuration and operational state for resource management.

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 June 18, 2016.

Copyright Notice

Table of Contents

1. Introduction

YANG [RFC6020] is a data definition language that was introduced to define the contents of a conceptual data store that allows networked devices to be managed using NETCONF[RFC6241]. YANG is proving relevant beyond its initial confines, as bindings to other interfaces(e.g. ReST) and encoding other than XML (e.g. JSON) are being defined. Furthermore, YANG data models can be used as the basis of implementation for other interface, such as CLI and Programmatic APIs.

The network device manages all kinds of resources such as memory which is one kind of system resources and MPLS label which is one type of typical service resources. The resource management is responsible for not only the creation and release of the pools of resources but also the allocation and release of the resources.

This document focuses on the service resources like MPLS label. The label space which is normally per-platform label space is shared by all the MPLS signalling protocols and applications. The labels used by mpls signalling protocols are allocated from the unified label space and released to the label space by the label management.

This document describes a YANG data model for resource management. The data model defines the configuration for pools of resources and operational state about the used and the available resources. Current version of the data model only includes MPLS label

management. Data model of other service resources will be defined in the future version.

2. Design of Data Model

2.1. Overview

The resource management in the network device is responsible for not only the creation and release of the pools of resources but also the allocation and release of the resources. The delimitation and the size of pools of resources can be system-defined or can be configured by operator. Normally the protocols and services will apply for the resources from resource management. Sometimes the operator uses the resources by configuration when the resources are not occupied. For example MPLS label is one kind of typical resource and when operator configures the transit static LSP manually the incoming label is needed to be configured. Therefore it’s necessary to provide the operational state of used and available resources for operator to select available resource to configure to avoid resource conflict configuration.

This document describes a YANG data model for resource management. The data model defines the configuration for pools of resources and operational state about the used and the available resources. Current version of the data model only includes MPLS label management.

The resource management YANG module is divided into two main containers :

o resource-cfg: that contains writable configuration objects about all kinds of resources.

o resource-state: that contains read-only operational state objects about all kinds of resources.

The figure below describes the overall structure of the resource management yang model :

```
module: ietf-resource-mgmt
   +--rw resource-cfg
   |  +--rw mpls-label-cfg
   |     +--rw label-space-cfg
   |        +--rw label-spaces* [label-type]
   |           +--rw label-type               label-type
   |           +--rw multiple-label-spaces* [label-begin label-end]
   |              +--rw label-begin    uint32
   |              +--rw label-end      uint32
   +--rw resource-state
      +--rw mpls-label-state
         +--ro label-space-state
         |  +--ro label-spaces* [label-type]
         |     +--ro label-type                      label-type
         |     +--ro label-total-number?             uint32
         |     +--ro label-available-total-number?   uint32
         |     +--ro multiple-label-spaces* [label-begin label-end]
         |        +--ro label-begin               uint32
         |        +--ro label-end                 uint32
         |        +--ro label-available-number?   uint32
         +--ro used-label
         |  +--ro used-labels* [label-value]
         |     +--ro label-value    uint32
         |     +--ro label-app?     label-app
         |     +--ro label-type?    label-type
         +--ro used-label-statistics
         |  +--ro used-label-statistics* [label-app]
         |     +--ro label-app           label-app
         |     +--ro used-label-count?   uint32
         +--ro available-label
            +--ro available-labels* [label-type]
               +--ro label-type        label-type
               +--ro label-segments* [label-begin label-end]
                  +--ro label-begin    uint32
                  +--ro label-end      uint32
```

2.2. Configuration

Resource-cfg container includes writable configuration objects about all kinds of resources like MPLS label.

2.2.1. Label Configuration

MPLS-label-cfg container defines the configuration objects about MPLS label. Label-space-cfg container defines the type of label space and a list of label spaces represented by a pair of lower-bound/upper-bound labels for each type of label space.

2.3. Operational State

Resource-state container includes read-only operational state objects about all kinds of resources like MPLS label.

2.3.1. Operational State of Label

MPLS-label-state container defines the operational state objects about MPLS label. It is divided into the following containers:

o label-space-state: it contains operational state object about label space which defines the type of label space and a list of label spaces represented by a pair of lower-bound/upper-bound labels for each type of label space and the statistics of number of available labels.

o used-label: it contains operational state object about used labels which defines the list of used labels which includes the used label value, what protocol or service uses the label and the type of label space which the label belongs to.

o used-label-statistics: it contains operational state object about used label statistics which defines the list of used label statistics of each protocol or service.

o available-label: it contains operational state object about available label which defines the type of label space and a list of available label spaces represented by a pair of lower-bound/upper-bound labels for each type of label space.

3. Yang Module

```
<CODE BEGINS> file "ietf-resource-mgmt@2015-12-16.yang"
module ietf-resource-mgmt{
  yang-version 1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-resource-mgmt";
  // replace with IANA namespace when assigned
  prefix "resource";

  organization "huawei";
  contact
    "lizhenbin@huawei.com
     jescia.chenxia@huawei.com";

  description "The yang data model defines the configuration and operation of re
source management.
               The initial version includes only label management.";

  revision "2015-12-16" {
```

```
    description "2015-10-15: Initial revision"+
                "2015-12-16: Add description for list and container";
    reference "TBD";
  }

  typedef label-type {
    type enumeration {
      enum static {
        description "The label is used for static configuration.";
      }
      enum dynamic {
        description "The label is used by dynamic mpls signalling protocol like
ldp, rsvp.";
      }
      enum block {
        description "The label is used by mpls protocol like kompella bgp with a
 block of labels allocated.";
      }
    }
    description
      "The type of label and the label space is partitioned according to the lab
el type.";
  }

  typedef label-app {
    type enumeration {
      enum p2p-rsvp-te {
        description "The label is allocated for RSVP-TE point-to-point lsp.";
      }
      enum p2p-ldp {
        description "The label is allocated for LDP point-to-point lsp.";
      }
      enum bgp {
        description "The label is allocated for labeled BGP(RFC3107).";
      }
      enum sr {
        description "The label is allocated for segment-routing.";
      }
      enum l2vpn {
        description "The label is allocated for l2vpn.";
      }
      enum l3vpn {
        description "The label is allocated for l3vpn.";
      }
      enum static-pw {
        description "The label is allocated for static pseudowire.";
      }
      enum static-lsp {
        description "The label is allocated for static lsp.";
      }
      enum mpls-tp-lsp {
```

```
        description "The label is allocated for static mpls tp lsp.";
      }
      enum p2mp-ldp {
        description "The label is allocated for LDP point-to-multipoint lsp.";
      }
      enum mp2mp-ldp {
        description "The label is allocated for LDP multipoint-to-multipoint lsp
.";
      }
    }
    description
      "The application which the label is allocated for.";
  }

  container resource-cfg{
    description "It includes writable configuration objects about all
                 kinds of resources. The initial version includes only
                 label management.";
    container mpls-label-cfg{
      description "It defines the configuration objects about mpls label.";
      container label-space-cfg{
        description "It defines the type of label space and a list of label
                     spaces.";
        list label-spaces {
          key "label-type";
          description "List of label spaces. Each label space defines for one ty
pe of label space.";

          leaf label-type {
            type label-type;
            description "The type of label and the label sapce is partitioned
                         according to the labe type.";
          }

          list multiple-label-spaces {
            key "label-begin label-end";
     description "List of discrete label spaces.";

            leaf label-begin {
              type uint32;
              description "The lower-bound label.";
            }

            leaf label-end {
              type uint32;
              description "The upper-bound label.";
            }
          }
        }
      }
```

```
    }
  }

  container resource-state{
    description "The configuration and operation of resource management.
                 The initial version includes only label management.";

    container mpls-label-state{
      description "The configuration and operation of mpls label management.";
      container label-space-state{
        config false;
        description "The label space.";

        list label-spaces {
          key "label-type";
          description "List of label spaces. Each label space defines for one ty
pe of label space.";

          leaf label-type {
            type label-type;
            description "The type of label and the label sapce is partitioned
                         according to the labe type.";
          }

          leaf label-total-number {
            type uint32;
            description "The total number of labels belonging to one type of
                         label space.";
          }

          leaf label-available-total-number {
            type uint32;
            description "The total number of available labels belonging to one
                         type of label space.";
          }

          list multiple-label-spaces {
            key "label-begin label-end";
            description "List of discrete label spaces.";

            leaf label-begin {
              type uint32;
              description "The lower-bound label.";
            }

            leaf label-end {
              type uint32;
              description "The upper-bound label.";
            }
```

```
            leaf label-available-number {
              type uint32;
              config false;
              description "The total number of available labels in one label
                           segment belonging to one type of label space.";
            }
          }
        }
      }

      container used-label{
        config false;
        description "Used labels.";

        list used-labels {
          key "label-value";
          description "List of used labels.";

          leaf label-value {
            type uint32;
            description "The value of the used label.";
          }

          leaf label-app {
            type label-app;
            description "The application which the label is allocated for.";
          }

          leaf label-type {
            type label-type;
            description "The type of label and the label sapce is partitioned
                         according to the labe type.";
          }
        }
      }

      container used-label-statistics{
        config false;
        description "The statistics of used labels for each application applying
 for labels.";

        list used-label-statistics {
          key "label-app";
          description "The list of the statistics of used labels.";

          leaf label-app {
            type label-app;
            description "The application which the label is allocated for.";
          }
```

```
          leaf used-label-count {
            type uint32;
            description "The count of used labels the special aplication occupie
s.";
          }
        }
      }

      container available-label{
        config false;
        description "Available labels.";

        list available-labels {
          key "label-type";
          description "List of available labels.";

          leaf label-type {
            type label-type;
            description "The type of label and the label sapce is partitioned
                         according to the labe type.";
          }

          list label-segments {
            key "label-begin label-end";
            description "List of label segments.";

            leaf label-begin {
              type uint32;
              description "The lower-bound label.";
            }

            leaf label-end {
              type uint32;
              description "The upper-bound label.";
            }
          }
        }
      }
    }
  }
}
<CODE ENDS>
```

4. IANA Considerations

This document registers the following URIs in the IETF XML registry [RFC3688]. Following the format in [RFC3688], the following registration is requested to be made.

URI: urn:ietf:params:xml:ns:yang:ietf-resource-mgmt XML: N/A, the requested URI is an XML namespace.

This document registers a YANG module in the YANG Module Names registry [RFC6020].

name: ietf-resource-mgmt namespace: urn:ietf:params:xml:ns:yang:ietf-resource-mgmt prefix: resource.

5. Security Considerations

The YANG module defined in this memo is designed to be accessed via the NETCONF protocol[RFC6241]. The lowest NETCONF layer is the secure transport layer and the mandatory-to-implement secure transport is SSH [RFC6242]. The NETCONF access control model [RFC6536] provides means to restrict access for particular NETCONF users to a pre-configured subset of all available NETCONF protocol operations and content. There are a number of data nodes defined in the YANG module which are writable/creatable/deletable (i.e., config true, which is the default). These data nodes may be considered sensitive or vulnerable in some network environments. Write operations (e.g., <edit-config>) to these data nodes without proper protection can have a negative effect on network operations.

6. Normative References

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

[RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <http://www.rfc-editor.org/info/rfc3688>.

[RFC6020] Bjorklund, M., Ed., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, DOI 10.17487/RFC6020, October 2010, <http://www.rfc-editor.org/info/rfc6020>.

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

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

[RFC6536] Bierman, A. and M. Bjorklund, "Network Configuration Protocol (NETCONF) Access Control Model", RFC 6536, DOI 10.17487/RFC6536, March 2012, <http://www.rfc-editor.org/info/rfc6536>.

Authors' Addresses

Xia Chen
Huawei Technologies
Huawei Bld., No.156 Beiqing Rd.
Beijing 100095
China

Email: jescia.chenxia@huawei.com

Zhenbin Li
Huawei Technologies
Huawei Bld., No.156 Beiqing Rd.
Beijing 100095
China

Email: lizhenbin@huawei.com

Routing area
Internet-Draft
Intended status: Standards Track
Expires: January 4, 2018

S. Hegde
Juniper Networks, Inc.
P. Sarkar
Individual

July 3, 2017

Micro-loop avoidance using SPRING
draft-hegde-rtgwg-microloop-avoidance-using-spring-03

Abstract

When there is a change in network topology either due to a link going down or due to a new link addition, all the nodes in the network need to get the complete view of the network and re-compute the routes. There will generally be a small time window when the forwarding state of each of the nodes is not synchronized. This can result in transient loops in the network, leading to dropped traffic due to over-subscription of links. Micro-looping is generally more harmful than simply dropping traffic on failed links, because it can cause control traffic to be dropped on an otherwise healthy link involved in micro-loop. This can lead to cascading adjacency failures or network meltdown.

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

Table of Contents

1. Introduction

Micro-loops are transient loops that occur during the period of time when some nodes have become aware of a topology change and have changed their forwarding tables in response, but slow routers have not yet modified their forwarding tables. This document provides

mechanisms to prevent micro-loops in the network in the event of link up/down or metric change.The micro-loop prevention mechanism uses the basic principles of near-side tunnelling as described in [RFC5715] sec 6.2.

Micro-loops can be formed involving the PLRs or nodes which are not directly connected to the link/node going down. The nodes which are not directly connected to the node/link going down/up are referred to as remote nodes. The micro-loop prevention mechanism described in this document prevents possible micro-loops involving the remote nodes. A new sub-tlv is defined in ISIS router capability TLV [RFC4971] and OSPF router capability TLV [RFC4970] for discovering support of this feature. The details are described in Section 4. The operational procedures for micro-loop prevention are described in Section 3.

2. Procedures for Micro-loop prevention

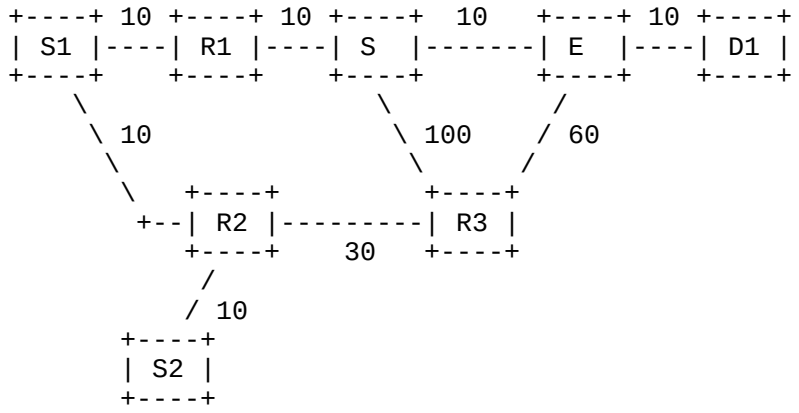


Figure 1: Sample Network

The topology shown in figure 1 illustrates a sample network topology where micro-loops can occur. The symmetric link metrics are shown in the diagram above. The traffic from S1 to D1 takes the path S1->R1->S->E->D1 and traffic from S2 takes the path S2->R2->S1->R1->S->E->D1 in normal operation. When the S->E link goes down, traffic can loop between S1->R2 when the FIB on S1 reflects the shortest path to D1 after the failure and the FIB on R2 reflects the shortest path to D1 before the failure. The mechanisms described in [I-D.ietf-rtgwg-uloop-delay] do not address micro-loops involving nodes that are not directly attached to the link that has just gone down or come up. For example when S->E link goes down, S

and E are the Point of Local Repair (PLR) and micro-loops formed between S1 and R2 are not handled.

The basic principle of the solution is to send the traffic on tunnelled paths for a certain time period until all the nodes in the network process the event and update their forwarding plane. When the link S->E goes down, all the nodes in the network tunnel the traffic to the nearest PLR. The PLR S needs to maintain the backup path created using FRR ([RFC5286]) or other mechanisms until all other nodes in the network converge. The PLR S forwards the traffic to the affected destinations via the back-up path until the convergence procedure is complete. This document assumes 100% backup coverage for the destinations via various FRR mechanisms. This document describes the procedures corresponding to the traffic flow from sources (S nodes) to the destination nodes (D nodes). The procedures equally apply to the D nodes being source and S nodes being destination.

As soon as a node learns of the topology change, it modifies its FIB to use loop-free tunnelled paths for the affected traffic, and it starts a "convergence delay timer". When the "convergence delay timer" expires, the node modifies its FIB to use the SPF path based on the changed topology. The use of tunnelled paths during the convergence period ensures that (barring other topology changes) all traffic affected by the topology change travels on a loop-free path.

After all the nodes in the network converge to actual SPF path,PLR converges to SPF path and updates the FIB. This micro-loop prevention mechanism delays the time it takes for routing to converge to the optimal paths in the new topology by a factor of 3 but the convergence time is deterministic and completely avoids micro-loops.

In principle, near-side tunnelling could be accomplished using labels distributed via LDP. However, since the application requires that any given router have the potential to create a tunnel to nearly every other router in the IGP domain, a large number of targeted LDP sessions would be needed to learn the FEC-label bindings distributed by the PLRs. SPRING [I-D.ietf-spring-segment-routing] provides a more efficient method for distributing shortest path labels for this application, since any router can compute the locally significant FEC-label bindings for any other router without the need for targeted LDP sessions.

[RFC5715] describes other mechanisms to prevent micro-loop prevention. Near-side tunnelling is more suited for deployments as it does not need additional computation or additional state maintenance in the network nodes.Far side tunnelling has the disadvantage that it requires the use of not-via addresses [RFC6981]

which requires additional address configuration on each node.Per destination non micro-looping path computation is another approach to prevent micro-loops but it is computationally intensive.

3. Detailed Solution based on SPRING

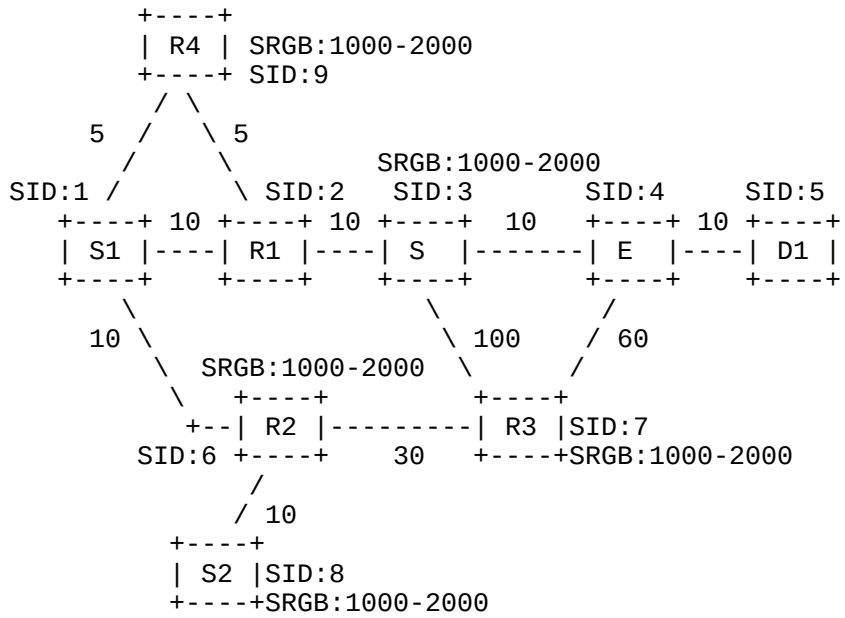


Figure 2: Sample SR Network

The above sample topology is provided with basic SPRING configurations of SRGB and the indices corresponding to each node. Each node has an SRGB 1000-2000 configured on the node. Same SRGB on all nodes is used for simplifying the example and the procedures are equally applicable when there is different SRGB configured on multiple nodes. Each node is provisioned with a MAX_CONVERGENCE_DELAY value that corresponds to its RIB to FIB convergence time. The information for support of the micro-loop prevention feature and the MAX_CONVERGENCE_DELAY value are flooded across the IGP domain (ISIS level/OSPF area). Each node in the IGP domain sets the MAX_CONVERGENCE_DELAY to the maximum of the values received in the domain.

3.1. Link-down event

When the S->E link goes down, all the nodes in the network receive the event via IGP database flooding. Each node supporting the micro-loop prevention mechanism specified in this document SHOULD perform the steps below.

1. The PLRs (S and E) perform FRR local repair for destinations affected by the failure of the link. Each computing node identifies the destinations affected by the topology change.In the example above, the destination D1 is affected by S->E link down for nodes S1,R1,R2, and R4. For S2, although the path to D1 changes there is no change in the immediate next-hop and hence its not necessary for S2 to perform any specific actions to prevent micro-loops.

2. For each affected destination, identify the nearest PLR advertising the change. The link-down event is advertised by both S and E. S is the nearest PLR for the nodes S1,R1,R2, and R4.

3. Let the S->E link down event occurs at time T0.

4. Start a timer T1 = max (all MAXIMUM_CONVERGENCE_DELAY) at all non-PLR nodes with affected destinations.

5. Start a timer T2 = 2 * T1 at the PLR.

6. For IP routes, modify the FIB for the affected destinations so that the nearest PLR’s node-sid is pushed on the packet’s label stack. For MPLS ingress and transit routes, modify the FIB for the affected destinations with a two label stack, the inner label corresponding to the destination and the outer label corresponding to the nearest PLR.

7. In the case of ECMP paths to the nearest PLR, both tunnelled paths are used. S1 has ECMP paths to the destination D1 and both the paths are impacted. Both the paths are modified to carry two label stacks containing the nearest PLR on top and the destination label at the bottom.

8. After the expiry of timer T1 all the non-PLR nodes modify their FIBs to use the shortest path as computed by the IGP, and they no longer push the node-SID of the nearest PLR on the packets.

9. After the expiry of T2, the PLR converges and updates the FIB to represent shortest path.

The ingress MPLS routes at various nodes for destination D1 at specified time intervals is mentioned below.

Node	Before T0	T0-T1	T1-T2	After T2
S1	Push 1005, Fwd to R1	Push 1005, 1003(top), Fwd to R1	Push 1005, Fwd to R2	Push 1005, Fwd to R2
	Push 1005, Fwd to R4	Push 1005, 1003(top), Fwd to R4		
S2	Push 1005, Fwd to R2	Push 1005, Fwd to R2	Push 1005, Fwd to R2	Push 1005, Fwd to R2
R1	Push 1005, Fwd to S	Push 1005, Fwd to S	Push 1005, Fwd to R4	Push 1005, Fwd to R4
			Push 1005, Fwd to S1	Push 1005, Fwd to S1
R2	Push 1005, Fwd to S1	Push 1005, 1003(top), Fwd to S1	Push 1005, Fwd to R3	Push 1005, Fwd to R3
R3	Push 1005, Fwd to E	Push 1005, 1003(top), Fwd to E	Push 1005, Fwd to E	Push 1005, Fwd to E
R4	Push 1005, Fwd to R1	Push 1005, 1003(top), Fwd to R1	Push 1005, Fwd to S1	Push 1005, Fwd to S1
S	Push 1005, Fwd to E	Push 1005, Fwd to R3 *	Push 1005, Fwd to R3 *	Push 1005, Fwd to R1
	Push 1005, Fwd to R3 *			Push 1005, Fwd to R3 *
E	Pop, Fwd to D1	Pop, Fwd to D1	Pop, Fwd to D1	Pop, Fwd to D1

* - Indicates backup path.

Figure 3: Sample MPLS ingress RIB

The corresponding MPLS transit routes at various nodes at specified time interval is shown below.

Node	Incoming Label	Before T0	T0-T1	T1-T2	After T2
S1	1005	Push 1005, Fwd to R1	Push 1005, 1003(top), Fwd to R1	Push 1005, Fwd to R2	Push 1005, Fwd to R2
		Push 1005, Fwd to R4	Push 1005, 1003(top), Fwd to R4		
	1003	Push 1003, Fwd to R1	Push 1003, Fwd to R1	Push 1003, Fwd to R2	Push 1003, Fwd to R2
S2	1005	Push 1005, Fwd to R2	Push 1005, Fwd to R2	Push 1005, Fwd to R2	Push 1005, Fwd to R2
	1003	Push 1003, Fwd to R1	Push 1003, Fwd to R1	Push 1003, Fwd to R2	Push 1003, Fwd to R2
R1	1005	Push 1005, Fwd to S	Push 1005, Fwd to S	Push 1005, Fwd to R4	Push 1005, Fwd to R4
				Push 1005, Fwd to S1	Push 1005, Fwd to S1
	1003	Push 1003, Fwd to S	Push 1003, Fwd to S	Push 1003, Fwd to S	Push 1003, Fwd to S
R2	1005	Push 1005, Fwd to	Push 1005, 1003(top), Fwd to S1	Push 1005, Fwd to R3	Push 1005, Fwd to R3

		S1			
	1003	Push 1003, Fwd to S1	Push 1003, Fwd to S1	Push 1003, Fwd to S1	Push 1003, Fwd to S1
R3	1005	Push 1005, Fwd to E	Push 1005, 1003(top), Fwd to E	Push 1005, Fwd to E	Push 1005, Fwd to E
	1003	Push 1003, Fwd to R2	Push 1003, Fwd to R2	Push 1003, Fwd to R2	Push 1003, Fwd to R2
R4	1005	Push 1005, Fwd to R1	Push 1005, 1003(top), Fwd to R1	Push 1005, Fwd to S1	Push 1005, Fwd to S1
	1003	Push 1003, Fwd to R1	Push 1003, Fwd to R1	Push 1003, Fwd to R1	Push 1003, Fwd to R1
S	1005	Push 1005, Fwd to E	Push 1005, Fwd to R3 *	Push 1005, Fwd to R3 *	Push 1005, Fwd to R1
		Push 1005, Fwd to R3 *			Push 1005, Fwd to R3 *
	1003	--	--	--	--
E	1005	Pop, Fwd to D1	Pop, Fwd to D1	Pop, Fwd to D1	Pop, Fwd to D1

* - Indicates backup path.

Figure 4: Sample MPLS transit RIB

3.2. Link-up event

When a new-link is added to the network, the PLR needs to update the FIB before it announces the change. First the PLR converges, updates the FIB as per the new-link based topology and then announces the new-link addition to the rest of the network. The other network nodes SHOULD follow the procedure exactly same as described in sec 3.1. They SHOULD update their FIB to tunnel the traffic to the closest node corresponding to the change.After MAX_CONVERGENCE_DELAY the nodes SHOULD update the FIB with the shortest path next-hops.

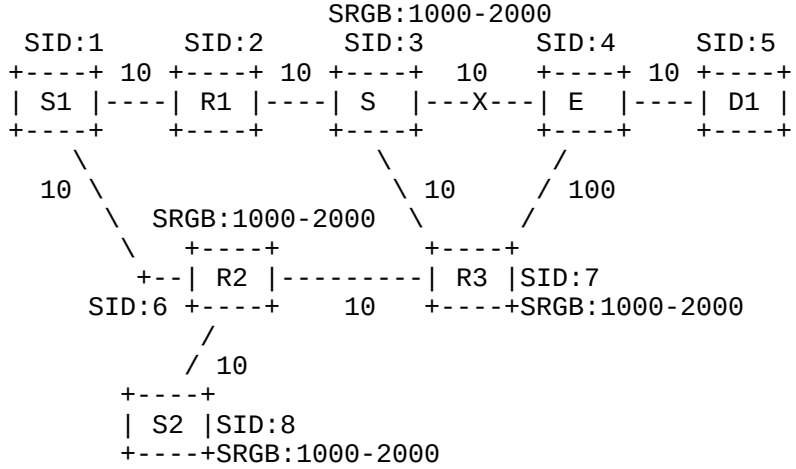


Figure 5: Sample SR Network

In the figure above, when the S->E link is added (or restored back),

1. PLR S processes the event and programs the FIB with new path for the affected destinations.

2. PLR delays flooding the event for MAX_CONVERGENCE_DELAY interval. This step prevents possible local micro-loop between S and R3.

3. Once PLR floods the event, non PLR nodes in the network identify the destinations affected by the database change. This is done by SPF computation and examining the next-hop change. The destination D1 is affected by S->E link up for nodes S1, R1, R2 and R3.

4. For each affected destination, identify the nearest PLR advertising the change. The link-up event is advertised by both

S and E. S is the nearest PLR for the nodes S1,R1,R2 and R3. When there are ECMP paths to the destination and a new ECMP path is added, the new ECMP path follows the micro-loop prevention mechanisms and tunnels the traffic towards nearest PLR.

5. Start a timer T3 = max (all MAXIMUM_CONVERGENCE_DELAY) at all non-PLR nodes.

6. For IP routes, update the FIB for the affected destinations so that the nearest PLR's node-sid is pushed on the packet's label stack. For MPLS ingress and transit router update the path with two label stack, the inner label corresponding to the destination and the outer label corresponding to the nearest PLR. This step prevents the possible remote micro-loop between S1 and R2.

7. After the expiry of timer T3 all the non-PLR nodes perform global convergence and update the FIB to represent the shortest path.

Other management events like metric change are handled similar to the link-down/link-up cases for metric increase/metric decrease cases respectively.

3.3. Computation of nearest PLR

When a network event is received by a node via the IGP database change notification, a node has to compute the nearest PLR corresponding to that advertisement. The first database change advertisement may be received from any of the PLRs, nearest or farthest.

3.3.1. Link down event

When a link goes down, IGPs generate a fresh LSP/Router LSA with the affected link removed. The computing node has to identify the missing link by walking over the LSP/LSA and compare the contents with an older version. Once the affected link is identified, the cost to reach both ends of the link should be examined. The nearest PLR is chosen based on the cost to reach the ends.

3.3.2. Node down event

When a node goes down, it is identified by the neighbouring nodes via link-down event. the neighbouring routers generate a fresh LSP/ Router LSA with the affected link removed. The computing node has to identify the missing link by walking over the LSP/LSA and compare the contents with an older version. Once the affected link is identified, the cost to reach both ends of the link should be

examined. The nearest PLR is chosen based on the cost to reach the ends.

When an advertisement from the farthest node is received before the nearest node, it is possible that the node that went down is chosen as the nearest PLR, as the node that went down might be still lingering in the database. In such cases node protection mechanisms for the deceased node at the previous-hop should prevent traffic loss. The details of such a mechanism is outside the scope of this document.

3.4. Handling multiple network events

It is important to categorize the received events as belonging to one network event or multiple network events. The link-down/link-up event is advertised by both ends of the link. The node-down/node-up event is advertised by all the neighbouring nodes.When an event is received, the computing node should analyse the changes in the database advertisements and compare with previous database.The micro-loop prevention procedures SHOULD be started when the first notification is received. The node SHOULD record the event for which micro-loop prevention procedures are being performed. If there are more database changes received during this time, the change should be mapped to the already on-going micro-loop prevention procedures.If the event is same then the micro-loop prevention procedures MUST continue, otherwise the micro-loop prevention procedures SHOULD be aborted.

[RFC5715] sec 6.2 describes mechanisms to handle the SRLG failures. If the received failure advertisement is part of an SRLG advertised in the IGP TE advertisement, the links on the path sharing same SRLG are identified and the tunnel is built with multiple label stack corresponding to the nearest PLR of each SRLG member.

When a failure is received, and the failure does not belong to the same SRLG as the already on-going micro-loop prevention, the micro-loop prevention procedures MUST be aborted and the normal convergence procedures SHOULD be followed.

3.4.1. Handling SRLG failures

Consider a sample network as shown above with S->E and S1->R1 belonging to same SRLG group. The symmetric link metrics are shown in the figure and the SRGB is 1000-2000 on all nodes. When the S->E link goes down, all the links belonging to the same SRLG are considered to be down and the route is modified to carry multiple node-sids along the path.

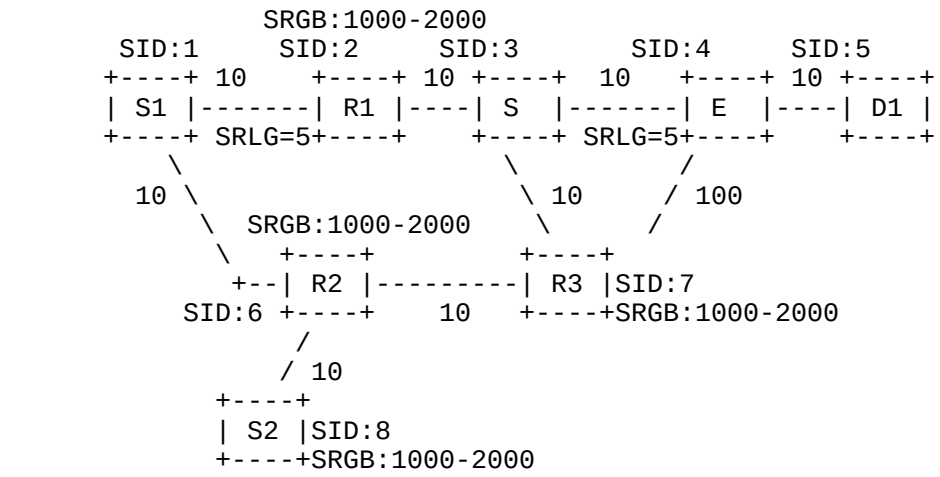


Figure 6: Sample Network with SRLG links

1. when the S->E link goes down, S and E generate the link down event, update their Router-LSA/ LSP and flood the updated information across the IGP domain.

2. The nodes in the IGP domain process the link-down event for affected destinations.If there are any other links with same SRLG on the path to destination, the nearest PLRs for those links are identified. In this example topology S1->R1 and S->E belong to same SRLG. For destination D1, R2 identifies two PLRs S1 and S for the S->E link down event.

3. The nodes build the tunnelled path having multiple labels for each of the identified links. for ex, R2 builds a stack containing node-sid of S1 and S. The tunnelled path at R2 looks as shown in Figure 7 below.

Node	Destination Prefix	Label Operation
R2	D1	Push 1005, 1003, 1001(top), Fwd to S1

Figure 7: Sample ingress RIB for SRLG failure handling

4. The procedures as described in sec 3.1 for the link-down event is followed to achieve micro-loop free convergence.

3.5. Handling ECMP

When a network event is received, if the the change causes only one of the ECMP paths to change, then the micro-loop prevention mechanisms described in sec 3.1 and 3.2 are applied to the changed path only. As described in section 3.1 and 3.2 , if there is an ECMP path to the nearest PLR, then all ECMP paths are used to tunnel the traffic during convergence.

3.6. Recognizing same network event

When a link goes down, both the ends of the link report the event by updating their LSP/LSA and flood it across the IGP domain. It is possible that the same network event being reported by two nodes is perceived as two different network events by the nodes in the IGP domain. The nodes processing the network events SHOULD evaluate if the received multiple events correspond to a single event by comparing the both ends of the reported link and also by looking at the previous event for which micro-loop prevention is being performed. If the event is same then micro-loop prevention procedures MUST be allowed to continue and MUST NOT be aborted.

Node down or new node addition events are reported by removing a link or adding a new link by all the adjacent nodes. In addition Node up event also comprises of a new LSA advertisement. The criteria to recognize if the event is same is to look at both ends of the changed link. If one end of the changed link maps to previously reported events and the other end of the link (advertising router) changes for each successive event, then the event is SHOULD be recognized as a new node addition or a node deletion. Micro-loop procedures MUST be allowed to continue and MUST NOT be aborted.

3.7. Partial deployment Considerations

The micro-loop mechanisms described in this document, are very effective and safe when all the nodes in the network support this feature and apply it when a network event happens. However, in some topologies, when all the nodes do not support the micro-loop prevention mechanism, the time duration of the loop can increase when only some nodes apply the procedures described in this document and some nodes do not.

For example, consider the sample topology described in the figure below.

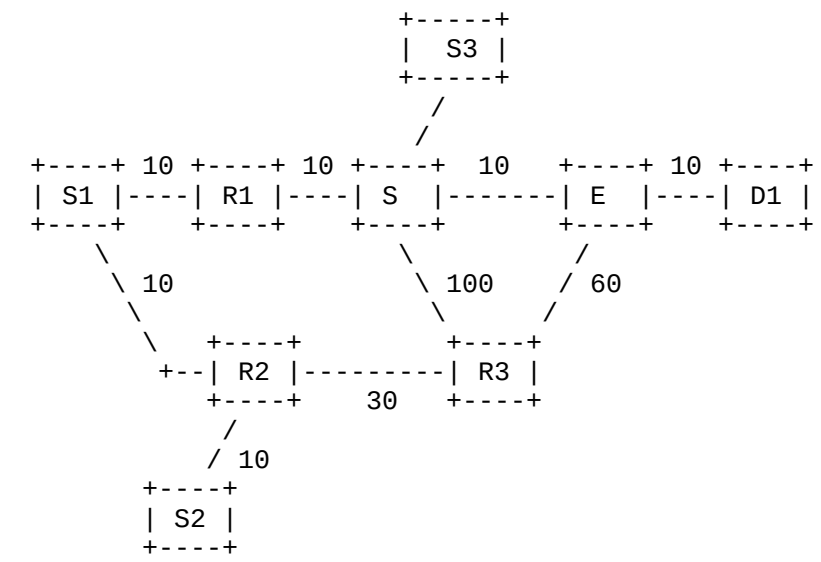


Figure 8: Sample Network with partial deployment

In this topology, S1, S2, and S3 are traffic sources and D1 is the destination. For each of the sources, Figure 9 shows the path before the failure (the before path) and the path after the failure (the post convergence path)..

Src	Dest	Original Path	Post-Convergence Path
S1	D1	S1->R1->S->E->D1	S1->R2->R3->E->D1
S2	D1	S2->R2->S1->R1->S->E->D1	S2->R2->R3->E->D1
S3	D1	S3->S->E->D1	S3->S->R1->S1->R2->R3->E->D1

Figure 9: Traffic flow in normal operation and post convergence path with S->E link down

In the above topology, if the PLR S does not support the micro-loop prevention mechanism but all other nodes support and apply this mechanism, then there is a possibility that the duration of traffic looping is higher than when the micro-loop prevention mechanisms are not applied at all. To mitigate this issue, protocol extensions to negotiate the support of this feature in the IGP domain is needed.

Section 4 describes the protocol mechanisms to advertise the support of this feature in OSPF and ISIS.

However, in certain deployments and topologies, it MAY be safe to apply the micro-loop prevention procedures even when all the nodes in the network do not support this feature, especially in topologies where the post convergence path from PLR does not traverse the nodes in P space of the PLR with respect to the the node or link being protected.

4. Protocol Procedures

4.1. OSPF

[RFC4970], defines Router Information (RI) LSA which may be used to advertise properties of the originating router. Payload of the RI LSA consists of one or more nested Type/Length/Value (TLV) triplets. This document defines a new TLV Micro-loop prevention support TLV which has following format:

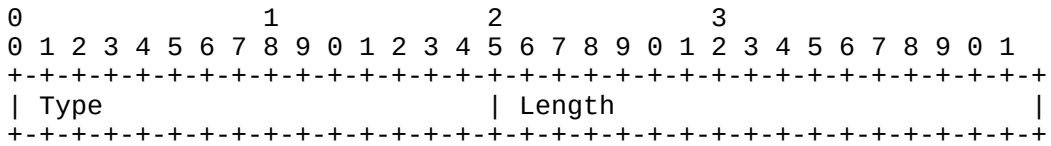


Figure 10: OSPF micro-loop prevention support TLV

Type : TBA, Suggested value 15

Length: 0

The MAX_CONVEREGENCE_DELAY described in this document is advertised using Controlled Convergence TLV as described in [I-D.ietf-ospf-mrt]

4.2. ISIS

[RFC4971], defines Router capability TLV which may be used to advertise properties of the originating router. This document defines a new sub-TLV Micro-loop prevention support sub-TLV which has following format:

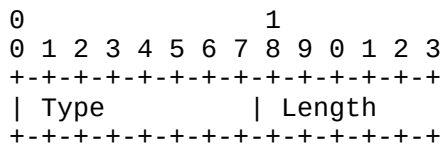


Figure 11: ISIS micro-loop prevention support sub-TLV

The Router Capability TLV specifies flags that control its advertisement. The Micro-loop prevention support sub-TLV MUST be propagated throughout the level and SHOULD NOT be advertised across level boundaries. Therefore Router Capability TLV distribution flags SHOULD be set accordingly, i.e.: the S flag in the Router Capability TLV [RFC4971] MUST be unset.

Type : TBA, Suggested value 5

Length: 0

The MAX_CONEVREGENCE_DELAY described in this document is advertised using Controlled Convergence TLV as described in [I-D.ietf-isis-mrt]

4.3. Elements of procedure

The micro-loop prevention support sub-TLV MUST be advertised only when the feature is enabled.When all the nodes in the IGP domain advertise this sub-TLV, a node supporting this feature MUST perform the micro-loop prevention procedures as described in this document. The micro-loop prevention mechanisms are applied within the OSPF area or ISIS level.

When there are one or more nodes in the IGP domain which do not support this feature, a node MAY perform micro-loop prevention procedures. Near side tunnelling mechanism ensures that when a group of nodes support this feature, traffic sourced from these set of nodes do not suffer micro-loop. A manageability interface SHOULD be provided to support micro-loop prevention in case of partial feature deployment.

5. Security Considerations

This document does not introduce any further security issues other than those discussed in [RFC2328] ,[RFC5340] , [ISO10589] and [RFC1195]

6. IANA Considerations

This specification updates one OSPF registry: OSPF Router Information (RI) TLVs Registry

i) TBD - Micro-loop prevention support TLV

This specification updates one ISIS registry: ISIS Router capability TLVs (TLV 242) Registry

i) TBD - Micro-loop prevention support sub-TLV

7. Acknowledgments

Thanks to Chris Bowers, Hannes Gredler,Eric Rosen and Stephane Litkowsky for valuable inputs.

8. References

8.1. Normative References

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

[RFC4970] Lindem, A., Ed., Shen, N., Vasseur, JP., Aggarwal, R., and S. Shaffer, "Extensions to OSPF for Advertising Optional Router Capabilities", RFC 4970, DOI 10.17487/RFC4970, July 2007, <http://www.rfc-editor.org/info/rfc4970>.

[RFC4971] Vasseur, JP., Ed., Shen, N., Ed., and R. Aggarwal, Ed., "Intermediate System to Intermediate System (IS-IS) Extensions for Advertising Router Information", RFC 4971, DOI 10.17487/RFC4971, July 2007, <http://www.rfc-editor.org/info/rfc4971>.

8.2. Informative References

[I-D.ietf-isis-mrt] Li, Z., Wu, N., Zhao, Q., Atlas, A., Bowers, C., and J. Tantsura, "Intermediate System to Intermediate System (IS-IS) Extensions for Maximally Redundant Trees (MRT)", draft-ietf-isis-mrt-03 (work in progress), June 2017.

[I-D.ietf-ospf-mrt]
Atlas, A., Hegde, S., Bowers, C., Tantsura, J., and Z. Li, "OSPF Extensions to Support Maximally Redundant Trees", draft-ietf-ospf-mrt-03 (work in progress), June 2017.

[I-D.ietf-rtgwg-uloop-delay]
Litkowski, S., Decraene, B., Filsfils, C., and P. Francois, "Micro-loop prevention by introducing a local convergence delay", draft-ietf-rtgwg-uloop-delay-05 (work in progress), June 2017.

[I-D.ietf-spring-segment-routing]
Filsfils, C., Previdi, S., Decraene, B., Litkowski, S., and R. Shakir, "Segment Routing Architecture", draft-ietf-spring-segment-routing-12 (work in progress), June 2017.

[ISO10589]
"Intermediate system to Intermediate system intra-domain routeing information exchange protocol for use in conjunction with the protocol for providing the connectionless-mode Network Service (ISO 8473), ISO/IEC 10589:2002, Second Edition.", Nov 2002.

[RFC1195] Callon, R., "Use of OSI IS-IS for routing in TCP/IP and dual environments", RFC 1195, DOI 10.17487/RFC1195, December 1990, <http://www.rfc-editor.org/info/rfc1195>.

[RFC2328] Moy, J., "OSPF Version 2", STD 54, RFC 2328, DOI 10.17487/RFC2328, April 1998, <http://www.rfc-editor.org/info/rfc2328>.

[RFC5286] Atlas, A., Ed. and A. Zinin, Ed., "Basic Specification for IP Fast Reroute: Loop-Free Alternates", RFC 5286, DOI 10.17487/RFC5286, September 2008, <http://www.rfc-editor.org/info/rfc5286>.

[RFC5340] Coltun, R., Ferguson, D., Moy, J., and A. Lindem, "OSPF for IPv6", RFC 5340, DOI 10.17487/RFC5340, July 2008, <http://www.rfc-editor.org/info/rfc5340>.

[RFC5715] Shand, M. and S. Bryant, "A Framework for Loop-Free Convergence", RFC 5715, DOI 10.17487/RFC5715, January 2010, <http://www.rfc-editor.org/info/rfc5715>.

[RFC6981] Bryant, S., Previdi, S., and M. Shand, "A Framework for IP and MPLS Fast Reroute Using Not-Via Addresses", RFC 6981, DOI 10.17487/RFC6981, August 2013, <http://www.rfc-editor.org/info/rfc6981>.

Authors' Addresses

Shraddha Hegde
Juniper Networks, Inc.
Exora Business Park
Bangalore, KA 560037
India

Email: shraddha@juniper.net

Pushpasis Sarkar
Individual

Email: pushpasis.ietf@gmail.com

RTGWG
Internet-Draft
Intended status: Standards Track
Expires: March 13, 2020

Y. Qu
Futurewei
J. Tantsura
Apstra
A. Lindem
Cisco
X. Liu
Volta Networks

September 10, 2019

A YANG Data Model for Routing Policy Management
draft-ietf-rtgwg-policy-model-07

Abstract

This document defines a YANG data model for configuring and managing routing policies in a vendor-neutral way and based on actual operational practice. The model provides a generic policy framework which can be augmented with protocol-specific policy configuration.

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 https://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 March 13, 2020.

Copyright Notice

Table of Contents

1. Introduction

This document describes a YANG [RFC6020] [RFC7950] data model for routing policy configuration based on operational usage and best practices in a variety of service provider networks. The model is intended to be vendor-neutral, in order to allow operators to manage policy configuration in a consistent, intuitive way in heterogeneous environments with routers supplied by multiple vendors.

The YANG modules in this document conform to the Network Management Datastore Architecture (NMDA) [RFC8342].

1.1. Goals and approach

This model does not aim to be feature complete -- it is a subset of the policy configuration parameters available in a variety of vendor implementations, but supports widely used constructs for managing how routes are imported, exported, and modified across different routing protocols. The model development approach has been to examine actual policy configurations in use across a number of operator networks. Hence the focus is on enabling policy configuration capabilities and structure that are in wide use.

Despite the differences in details of policy expressions and conventions in various vendor implementations, the model reflects the observation that a relatively simple condition-action approach can be readily mapped to several existing vendor implementations, and also gives operators an intuitive and straightforward way to express policy without sacrificing flexibility. A side affect of this design decision is that legacy methods for expressing policies are not considered. Such methods could be added as an augmentation to the model if needed.

Consistent with the goal to produce a data model that is vendor neutral, only policy expressions that are deemed to be widely available in existing major implementations are included in the model. Those configuration items that are only available from a single implementation are omitted from the model with the expectation they will be available in separate vendor-provided modules that augment the current model.

2. Terminology and Notation

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

The following terms are defined in [RFC8342]:

- client
- server
- configuration
- system state
- operational state

- intended configuration

The following terms are defined in [RFC7950]:

- action
- augment
- container
- container with presence
- data model
- data node
- feature
- leaf
- list
- mandatory node
- module
- schema tree
- RPC (Remote Procedure Call) operation

2.1. Tree Diagrams

Tree diagrams used in this document follow the notation defined in [RFC8340].

2.2. Prefixes in Data Node Names

In this document, names of data nodes, actions, and other data model objects are often used without a prefix, as long as it is clear from the context in which YANG module each name is defined. Otherwise, names are prefixed using the standard prefix associated with the corresponding YANG module, as shown in Table 1.

Prefix	YANG module	Reference
if	ietf-interfaces	[RFC8343]
rt	ietf-routing	[RFC8349]
yang	ietf-yang-types	[RFC6991]
inet	ietf-inet-types	[RFC6991]
if-cmn	ietf-interfaces-common	[I-D.ietf-netmod-intf-ext-yang]

Table 1: Prefixes and Corresponding YANG Modules

3. Model overview

The routing policy module has three main parts:

- A generic framework to express policies as sets of related conditions and actions. This includes match sets and actions that are useful across many routing protocols.

- A structure that allows routing protocol models to add protocol-specific policy conditions and actions though YANG augmentations. There is a complete example of this for BGP [RFC4271] policies in the proposed vendor-neutral BGP data model [I-D.ietf-idr-bgp-model].

- A reusable grouping for attaching import and export rules in the context of routing configuration for different protocols, VRFs, etc. This also enables creation of policy chains and expressing default policy behavior.

The module makes use of the standard Internet types, such as IP addresses, autonomous system numbers, etc., defined in RFC 6991 [RFC6991].

4. Route policy expression

Policies are expressed as a sequence of top-level policy definitions each of which consists of a sequence of policy statements. Policy statements in turn consist of simple condition-action tuples. Conditions may include multiple match or comparison operations, and similarly, actions may effect multiple changes to route attributes, or indicate a final disposition of accepting or rejecting the route. This structure is shown below.

```
+--rw routing-policy
   +--rw policy-definitions
      +--rw policy-definition* [name]
         +--rw name          string
         +--rw statements
            +--rw statement* [name]
               +--rw name          string
               +--rw conditions
               |     ...
               +--rw actions
                     ...
```

4.1. Defined sets for policy matching

The models provides a set of generic sets that can be used for matching in policy conditions. These sets are applicable for route selection across multiple routing protocols. They may be further augmented by protocol-specific models which have their own defined sets. The supported defined sets include:

- prefix sets - define a set of IP prefixes, each with an associated CIDR netmask range (or exact length)

- neighbor sets - define a set of neighboring nodes by their IP addresses. These sets are used for selecting routes based on the neighbors advertising the routes.

- tag set - define a set of generic tag values that can be used in matches for filtering routes

The model structure for defined sets is shown below.

```
      +--rw routing-policy
         +--rw defined-sets
         |  +--rw prefix-sets
         |  |  +--rw prefix-set* [name]
         |  |     +--rw name        string
         |  |     +--rw mode?       enumeration
         |  |     +--rw prefixes
         |  |        +--rw prefix-list* [ip-prefix masklength-lower
         |  |                            masklength-upper]
         |  |           +--rw ip-prefix           inet:ip-prefix
         |  |           +--rw masklength-lower    uint8
         |  |           +--rw masklength-upper    uint8
         |  +--rw neighbor-sets
         |  |  +--rw neighbor-set* [name]
         |  |     +--rw name       string
         |  |     +--rw address*   inet:ip-address
         |  +--rw tag-sets
         |     +--rw tag-set* [name]
         |        +--rw name         string
         |        +--rw tag-value*   tag-type
```

4.2. Policy conditions

Policy statements consist of a set of conditions and actions (either of which may be empty). Conditions are used to match route attributes against a defined set (e.g., a prefix set), or to compare attributes against a specific value.

Match conditions may be further modified using the match-set-options configuration which allows operators to change the behavior of a match. Three options are supported:

- ALL - match is true only if the given value matches all members of the set.
- ANY - match is true if the given value matches any member of the set.
- INVERT - match is true if the given value does not match any member of the given set.

Not all options are appropriate for matching against all defined sets (e.g., match ALL in a prefix set does not make sense). In the model, a restricted set of match options is used where applicable.

Comparison conditions may similarly use options to change how route attributes should be tested, e.g., for equality or inequality, against a given value.

While most policy conditions will be added by individual routing protocol models via augmentation, this routing policy model includes several generic match conditions and also the ability to test which protocol or mechanism installed a route (e.g., BGP, IGP, static, etc.). The conditions included in the model are shown below.

```
+--rw routing-policy
  +--rw policy-definitions
      +--rw policy-definition* [name]
         +--rw name          string
         +--rw statements
            +--rw statement* [name]
               +--rw conditions
                  |  +--rw call-policy?
                  |  +--rw install-protocol-eq?
                  |  +--rw match-interface
                  |  |  +--rw interface?
                  |  |  +--rw subinterface?
                  |  +--rw match-prefix-set
                  |  |  +--rw prefix-set?
                  |  |  +--rw match-set-options?
                  |  +--rw match-neighbor-set
                  |  |  +--rw neighbor-set?
                  |  +--rw match-tag-set
                  |     +--rw tag-set?
                  |     +--rw match-set-options?
```

4.3. Policy actions

When policy conditions are satisfied, policy actions are used to set various attributes of the route being processed, or to indicate the final disposition of the route, i.e., accept or reject.

Similar to policy conditions, the routing policy model includes generic actions in addition to the basic route disposition actions. These are shown below.

```
      +--rw routing-policy
        +--rw policy-definitions
           +--rw policy-definition* [name]
              +--rw statements
                 +--rw statement* [name]
                    +--rw actions
                      +--rw policy-result?    policy-result-type
                      +--rw set-metric?       uint32
                      +--rw set-preference?   uint16
```

4.4. Policy subroutines

Policy 'subroutines' (or nested policies) are supported by allowing policy statement conditions to reference other policy definitions using the call-policy configuration. Called policies apply their conditions and actions before returning to the calling policy statement and resuming evaluation. The outcome of the called policy affects the evaluation of the calling policy. If the called policy results in an accept-route, then the subroutine returns an effective boolean true value to the calling policy. For the calling policy, this is equivalent to a condition statement evaluating to a true value and evaluation of the policy continues (see Section 5). Note that the called policy may also modify attributes of the route in its action statements. Similarly, a reject-route action returns false and the calling policy evaluation will be affected accordingly. When the end of the subroutine policy chain is reached, the default route disposition action is returned (i.e., boolean false for reject-route unless an alternate default action is specified for the chain). Consequently, a subroutine cannot explicitly accept or reject a route. Rather it merely provides an indication that 'call-policy' condition returns boolean true or false indicating whether or not the condition matches. Route acceptance or rejection is solely determined by the top-level policy.

Note that the called policy may itself call other policies (subject to implementation limitations). The model does not prescribe a nesting depth because this varies among implementations. For example, some major implementation may only support a single level of subroutine recursion. As with any routing policy construction, care must be taken with nested policies to ensure that the effective return value results in the intended behavior. Nested policies are a convenience in many routing policy constructions but creating policies nested beyond a small number of levels (e.g., 2-3) should be discouraged.

5. Policy evaluation

Evaluation of each policy definition proceeds by evaluating its corresponding individual policy statements in order. When all the condition statements in a policy statement are satisfied, the corresponding action statements are executed. If the actions include either accept-route or reject-route actions, evaluation of the current policy definition stops, and no further policy definitions in the chain are evaluated.

If the conditions are not satisfied, then evaluation proceeds to the next policy statement. If none of the policy statement conditions are satisfied, then evaluation of the current policy definition stops, and the next policy definition in the chain is evaluated. When the end of the policy chain is reached, the default route disposition action is performed (i.e., reject-route unless an alternate default action is specified for the chain).

Note that the route's pre-policy attributes are always used for testing policy statement conditions. In other words, if actions modify the policy application specific attributes, those modifications are not used for policy statement conditions.

6. Applying routing policy

Routing policy is applied by defining and attaching policy chains in various routing contexts. Policy chains are sequences of policy definitions (described in Section 4) that have an associated direction (import or export) with respect to the routing context in which they are defined. The routing policy model defines an apply-policy grouping that can be imported and used by other models. As shown below, it allows definition of import and export policy chains, as well as specifying the default route disposition to be used when no policy definition in the chain results in a final decision.

```
+--rw apply-policy
|  +--rw import-policy*
|  +--rw default-import-policy?   default-policy-type
|  +--rw export-policy*
|  +--rw default-export-policy?   default-policy-type
```

The default policy defined by the model is to reject the route for both import and export policies.

7. Routing protocol-specific policies

Routing models that require the ability to apply routing policy may augment the routing policy model with protocol or other specific policy configuration. The routing policy model assumes that additional defined sets, conditions, and actions may all be added by other models.

An example of this is shown below, in which the BGP configuration model in [I-D.ietf-idr-bgp-model] adds new defined sets to match on community values or AS paths. The model similarly augments BGP-specific conditions and actions in the corresponding sections of the routing policy model.

```
+--rw routing-policy
   +--rw defined-sets
   |  +--rw prefix-sets
   |  |  +--rw prefix-set* [name]
   |  |     +--rw name        string
   |  |     +--rw mode?       enumeration
   |  |     +--rw prefixes
   |  |        +--rw prefix-list* [ip-prefix masklength-lower
   |  |                            masklength-upper]
   |  |           +--rw ip-prefix           inet:ip-prefix
   |  |           +--rw masklength-lower    uint8
   |  |           +--rw masklength-upper    uint8
   |  +--rw neighbor-sets
   |  |  +--rw neighbor-set* [name]
   |  |     +--rw name       string
   |  |     +--rw address*   inet:ip-address
   |  +--rw tag-sets
   |  |  +--rw tag-set* [name]
   |  |     +--rw name         string
   |  |     +--rw tag-value*   tag-type
   |  +--rw bgp-pol:bgp-defined-sets
   |     +--rw bgp-pol:community-sets
   |     |  +--rw bgp-pol:community-set* [community-set-name]
   |     |     +--rw bgp-pol:community-set-name    string
   |     |     +--rw bgp-pol:community-member*     union
   |     +--rw bgp-pol:ext-community-sets
   |     |  +--rw bgp-pol:ext-community-set* [ext-community-set-name]
   |     |     +--rw bgp-pol:ext-community-set-name    string
   |     |     +--rw bgp-pol:ext-community-member*     union
   |     +--rw bgp-pol:as-path-sets
   |        +--rw bgp-pol:as-path-set* [as-path-set-name]
   |           +--rw bgp-pol:as-path-set-name      string
   |           +--rw bgp-pol:as-path-set-member*   string
   +--rw policy-definitions
```

```
         +--rw policy-definition* [name]
            +--rw name          string
            +--rw statements
               +--rw statement* [name]
                  +--rw name          string
                  +--rw conditions
                  |  +--rw call-policy?
                  |  +--rw source-protocol?          identityref
                  |  +--rw match-interface
                  |  |  +--rw interface?
                  |  |  +--rw subinterface?
                  |  +--rw match-prefix-set
                  |  |  +--rw prefix-set?
                  |  |  +--rw match-set-options? match-set-options-type
                  |  +--rw match-neighbor-set
                  |  |  +--rw neighbor-set?
                  |  +--rw match-tag-set
                  |  |  +--rw tag-set?
                  |  |  +--rw match-set-options? match-set-options-type
                  |  +--rw bgp-pol:bgp-conditions
                  |     +--rw bgp-pol:med-eq?      uint32
                  |     +--rw bgp-pol:origin-eq?
                  |           bgp-types:bgp-origin-attr-type
                  |     +--rw bgp-pol:next-hop-in*
                  |           inet:ip-address-no-zone
                  |     +--rw bgp-pol:afi-safi-in*   identityref
                  |     +--rw bgp-pol:local-pref-eq? uint32
                  |     +--rw bgp-pol:route-type?    enumeration
                  |     +--rw bgp-pol:community-count
                  |     +--rw bgp-pol:as-path-length
                  |     +--rw bgp-pol:match-community-set
                  |     |  +--rw bgp-pol:community-set?
                  |     |  +--rw bgp-pol:match-set-options?
                  |     |        match-set-options-type
                  |     +--rw bgp-pol:match-ext-community-set
                  |     |  +--rw bgp-pol:ext-community-set?
                  |     |  +--rw bgp-pol:match-set-options?
                  |     |        match-set-options-type
                  |     +--rw bgp-pol:match-as-path-set
                  |        +--rw bgp-pol:as-path-set?
                  |        +--rw bgp-pol:match-set-options?
                  |              match-set-options-type
                  +--rw actions
                     +--rw policy-result?         policy-result-type
                     +--rw set-metric?            uint32
                     +--rw set-preference?        uint16
                     +--rw bgp-pol:bgp-actions
                        +--rw bgp-pol:set-route-origin?
```

```
                        bgp-types:bgp-origin-attr-type
                     +--rw bgp-pol:set-local-pref?        uint32
                     +--rw bgp-pol:set-next-hop?   bgp-next-hop-type
                     +--rw bgp-pol:set-med?        bgp-set-med-type
                     +--rw bgp-pol:set-as-path-prepend
                     |  +--rw bgp-pol:repeat-n?   uint8
                     +--rw bgp-pol:set-community
                     |  +--rw bgp-pol:method?      enumeration
                     |  +--rw bgp-pol:options?
                     |        bgp-set-community-option-type
                     |  +--rw bgp-pol:inline
                     |  |  +--rw bgp-pol:communities*   union
                     |  +--rw bgp-pol:reference
                     |     +--rw bgp-pol:community-set-ref?
                     +--rw bgp-pol:set-ext-community
                        +--rw bgp-pol:method?       enumeration
                        +--rw bgp-pol:options?
                              bgp-set-community-option-type
                        +--rw bgp-pol:inline
                        |  +--rw bgp-pol:communities*   union
                        +--rw bgp-pol:reference
                           +--rw bgp-pol:ext-community-set-ref?
```

8. Security Considerations

Routing policy configuration has a significant impact on network operations, and, as such, any related model carries potential security risks.

YANG data models are generally designed to be used with the NETCONF protocol over an SSH transport. This provides an authenticated and secure channel over which to transfer configuration and operational data. Note that use of alternate transport or data encoding (e.g., JSON over HTTPS) would require similar mechanisms for authenticating and securing access to configuration data.

Most of the data elements in the policy model could be considered sensitive from a security standpoint. Unauthorized access or invalid data could cause major disruption.

9. IANA Considerations

This YANG data model and the component modules currently use a temporary ad-hoc namespace. If and when it is placed on redirected for the standards track, an appropriate namespace URI will be registered in the IETF XML Registry" [RFC3688]. The routing policy YANG modules will be registered in the "YANG Module Names" registry [RFC6020].

10. YANG modules

The routing policy model is described by the YANG modules in the sections below.

10.1. Routing policy model

```
<CODE BEGINS> file "ietf-routing-policy@2019-03-06.yang"
module ietf-routing-policy {

  yang-version "1.1";
  namespace "urn:ietf:params:xml:ns:yang:ietf-routing-policy";
  prefix rt-pol;

  import ietf-inet-types {
    prefix "inet";
  }

  import ietf-yang-types {
    prefix "yang";
  }

  import ietf-interfaces {
    prefix "if";
  }

  import ietf-routing {
    prefix "rt";
  }

  import ietf-interfaces-common {
    prefix if-cmn;
  }

  import ietf-if-l3-vlan {
    prefix "if-l3-vlan";
  }


 organization
    "IETF RTGWG - Routing Area Working Group";
  contact
    "WG Web:   <http://tools.ietf.org/wg/rtgwg/>
     WG List:  <mailto:rtgwg@ietf.org>


     Editor:   Yingzhen Qu
               <mailto:yingzhen.qu@huawei.com>
```

Jeff Tantsura
<mailto:jefftant.ietf@gmail.com>
Acee Lindem
<mailto:acee@cisco.com>
Xufeng Liu
<mailto:xufeng_liu@jabil.com>
Anees Shaikh
<mailto:aashaikh@google.com>";

```
description
  "This module describes a YANG model for routing policy
   configuration. It is a limited subset of all of the policy
   configuration parameters available in the variety of vendor
   implementations, but supports widely used constructs for
   managing how routes are imported, exported, and modified across
   different routing protocols.  This module is intended to be
   used in conjunction with routing protocol configuration modules
   (e.g., BGP) defined in other models.

   Route policy expression:

   Policies are expressed as a set of top-level policy
   definitions, each of which consists of a sequence of policy
   statements. Policy statements consist of simple
   condition-action tuples. Conditions may include mutiple match
   or comparison operations, and similarly actions may be
   multitude of changes to route attributes or a final disposition
   of accepting or rejecting the route.

   Route policy evaluation:

   Policy definitions are referenced in routing protocol
   configurations using import and export configuration
   statements. The arguments are members of an ordered list of
   named policy definitions which comprise a policy chain, and
   optionally, an explicit default policy action (i.e., reject
   or accept).

   Evaluation of each policy definition proceeds by evaluating its
   corresponding individual policy statements in order.  When a
   condition statement in a policy statement is satisfied, the
   corresponding action statement is executed.  If the action
   statement has either accept-route or reject-route actions,
   policy evaluation of the current policy definition stops, and
   no further policy definitions in the chain are evaluated.

   If the condition is not satisfied, then evaluation proceeds to
   the next policy statement.  If none of the policy statement
```

```
        conditions are satisfied, then evaluation of the current policy
        definition stops, and the next policy definition in the chain
        is evaluated.  When the end of the policy chain is reached, the
        default route disposition action is performed (i.e.,
        reject-route unless an alternate default action is specified
        for the chain).

        Policy 'subroutines' (or nested policies) are supported by
        allowing policy statement conditions to reference another
        policy definition which applies conditions and actions from
        the referenced policy before returning to the calling policy
        statement and resuming evaluation.  If the called policy
        results in an accept-route (either explicit or by default),
        then the subroutine returns an effective true value to the
        calling policy.  Similarly, a reject-route action returns
        false.  If the subroutine returns true, the calling policy
        continues to evaluate the remaining conditions (using a
        modified route if the subroutine performed any changes to the
        route).";

    revision "2019-03-06" {
      description
        "Initial revision.";
      reference
       "RFC XXXX: Routing Policy Configuration Model for Service
        Provider Networks";
    }

    // typedef statements

    typedef default-policy-type {
      // this typedef retained for name compatibiity with default
      // import and export policy
      type enumeration {
        enum accept-route {
          description
            "Default policy to accept the route";
        }
        enum reject-route {
          description
            "Default policy to reject the route";
        }
      }
      description
        "Type used to specify route disposition in
         a policy chain";
```

```
    }

    typedef policy-result-type {
      type enumeration {
        enum accept-route {
          description "Policy accepts the route";
        }
        enum reject-route {
          description "Policy rejects the route";
        }
      }
      description
        "Type used to specify route disposition in
         a policy chain";
    }

    typedef tag-type {
      type union {
        type uint32;
        type yang:hex-string;
      }
      description "Type for expressing route tags on a local system,
         including IS-IS and OSPF; may be expressed as either decimal
         or hexadecimal integer";
      reference
        "RFC 2178 - OSPF Version 2
         RFC 5130 - A Policy Control Mechanism in IS-IS Using
                    Administrative Tags";
    }

    typedef match-set-options-type {
      type enumeration {
        enum any {
          description "Match is true if given value matches any member
             of the defined set";
        }
        enum all {
          description "Match is true if given value matches all
             members of the defined set";
        }
        enum invert {
          description "Match is true if given value does not match any
             member of the defined set";
        }
      }
      default any;
      description
        "Options that govern the behavior of a match statement.  The
```

```
          default behavior is any, i.e., the given value matches any
          of the members of the defined set";
      }


      // grouping statements

      grouping prefix-set {
        description
          "Configuration data for prefix sets used in policy
           definitions.";

        leaf name {
          type string;
          description
            "Name of the prefix set -- this is used as a label to
             reference the set in match conditions";
        }

        leaf mode {
          type enumeration {
            enum ipv4 {
              description
                "Prefix set contains IPv4 prefixes only";
            }
            enum ipv6 {
              description
                "Prefix set contains IPv6 prefixes only";
            }
            enum mixed {
              description
                "Prefix set contains mixed IPv4 and IPv6 prefixes";
            }
          }
          description
            "Indicates the mode of the prefix set, in terms of which
             address families (IPv4, IPv6, or both) are present.  The
             mode provides a hint, but the device must validate that all
             prefixes are of the indicated type, and is expected to
             reject the configuration if there is a discrepancy.  The
             MIXED mode may not be supported on devices that require
             prefix sets to be of only one address family.";
        }

      }

      grouping prefix-set-top {
        description
```

```
        "Top-level data definitions for a list of IPv4 or IPv6
         prefixes which are matched as part of a policy";

      container prefix-sets {
        description
          "Enclosing container ";

        list prefix-set {
          key "name";
          description
            "List of the defined prefix sets";

          uses prefix-set;

          uses prefix-top;
        }
      }
    }

    grouping prefix {
      description
        "Configuration data for a prefix definition";

      leaf ip-prefix {
        type inet:ip-prefix;
        mandatory true;
        description
          "The prefix member in CIDR notation -- while the
           prefix may be either IPv4 or IPv6, most
           implementations require all members of the prefix set
           to be the same address family.  Mixing address types in
           the same prefix set is likely to cause an error.";
      }

      leaf masklength-lower {
        type uint8;
        description
          "Masklength range lower bound.";
      }
      leaf masklength-upper {
        type uint8 {
          range "1..128";
        }
        must "../masklength-upper >= ../masklength-lower" {
          error-message "The upper bound should not be less"
                      + "than lower bound.";
        }
        description
```

```
          "Masklength range upper bound.

           The combination of masklength-lower and masklength-upper
           define a range for the mask length, or single 'exact'
           length if masklength-lower and masklenght-upper are equal.

           Example: 10.3.192.0/21 through 10.3.192.0/24 would be
           expressed as prefix: 10.3.192.0/21,
                        masklength-lower=21,
                        masklength-upper=24

           Example: 10.3.192.0/21 (an exact match) would be
           expressed as prefix: 10.3.192.0/21,
                        masklength-lower=21,
                        masklength-upper=21";
      }
    }

    grouping prefix-top {
      description
        "Top-level grouping for prefixes in a prefix list";

      container prefixes {
        description
          "Enclosing container for the list of prefixes in a policy
           prefix list";

        list prefix-list {
          key "ip-prefix masklength-lower masklength-upper";
          description
            "List of prefixes in the prefix set";

          uses prefix;
        }
      }
    }

    grouping neighbor-set {
      description
        "This grouping provides neighbor set definitions";

      leaf name {
        type string;
        description
            "Name of the neighbor set -- this is used as a label
             to reference the set in match conditions";
      }
```

```
      leaf-list address {
        type inet:ip-address;
        description
          "List of IP addresses in the neighbor set";
      }
    }

    grouping neighbor-set-top {
      description
        "Top-level data definition for a list of IPv4 or IPv6
         neighbors which can be matched in a routing policy";

      container neighbor-sets {
        description
          "Enclosing container for the list of neighbor set
           definitions";

        list neighbor-set {
          key "name";
          description
            "List of defined neighbor sets for use in policies.";

          uses neighbor-set;
        }
      }
    }

    grouping tag-set {
      description
        "This grouping provides tag set definitions.";

      leaf name {
        type string;
        description
          "Name of the tag set -- this is used as a label to reference
           the set in match conditions";
      }

      leaf-list tag-value {
        type tag-type;
        description
          "Value of the tag set member";
      }
    }

    grouping tag-set-top {
      description
        "Top-level data definitions for a list of tags which can
```

```
          be matched in policies";

        container tag-sets {
          description
            "Enclosing container for the list of tag sets.";

          list tag-set {
            key "name";
            description
              "List of tag set definitions.";

            uses tag-set;

          }
        }
      }


      grouping match-set-options-group {
        description
          "Grouping containing options relating to how a particular set
           should be matched";

        leaf match-set-options {
          type match-set-options-type;
          description
            "Optional parameter that governs the behavior of the
             match operation";
        }
      }

      grouping match-set-options-restricted-group {
        description
          "Grouping for a restricted set of match operation modifiers";

        leaf match-set-options {
          type match-set-options-type {
            enum any {
              description "Match is true if given value matches any
                 member of the defined set";
            }
            enum invert {
              description "Match is true if given value does not match
                 any member of the defined set";
            }
          }
          description
            "Optional parameter that governs the behavior of the
```

```
            match operation.  This leaf only supports matching on ANY
            member of the set or inverting the match.  Matching on ALL
            is not supported";
        }
      }


      grouping match-interface-condition {
        description
          "This grouping provides interface match condition";

        container match-interface {
          leaf interface {
            type leafref {
              path "/if:interfaces/if:interface/if:name";
            }
            description
              "Reference to a base interface.  If a reference to a
               subinterface is required, this leaf must be specified
               to indicate the base interface.";
          }
          leaf subinterface {
            type leafref {
              path "/if:interfaces/if:interface/if-cmn:encapsulation"
                 + "/if-l3-vlan:dot1q-vlan"
                 + "/if-l3-vlan:outer-tag/if-l3-vlan:vlan-id";
            }
            description
              "Reference to a subinterface -- this requires the base
               interface to be specified using the interface leaf in
               this container.  If only a reference to a base interface
               is requuired, this leaf should not be set.";
          }

          description
            "Container for interface match conditions";
        }
      }

      grouping prefix-set-condition {
        description
          "This grouping provides prefix-set conditions";

        container match-prefix-set {
          leaf prefix-set {
            type leafref {
              path "../../../../../../../defined-sets/" +
```

```
              "prefix-sets/prefix-set/name";
          }
          description "References a defined prefix set";
        }
        uses match-set-options-restricted-group;

        description
          "Match a referenced prefix-set according to the logic
           defined in the match-set-options leaf";
      }
    }

    grouping neighbor-set-condition {
      description
        "This grouping provides neighbor-set conditions";

      container match-neighbor-set {
        leaf neighbor-set {
          type leafref {
            path "../../../../../../../defined-sets/neighbor-sets/" +
            "neighbor-set/name";
            require-instance true;
          }
          description "References a defined neighbor set";
        }

        description
          "Match a referenced neighbor set according to the logic
           defined in the match-set-options-leaf";
      }
    }

    grouping tag-set-condition {
      description
        "This grouping provides tag-set conditions";

      container match-tag-set {
        leaf tag-set {
          type leafref {
            path  "../../../../../../../defined-sets/tag-sets" +
            "/tag-set/name";
            require-instance true;
          }
          description "References a defined tag set";
        }
        uses match-set-options-restricted-group;

        description
```

```
          "Match a referenced tag set according to the logic defined
           in the match-options-set leaf";
      }
    }

    grouping generic-conditions {
      description "Condition statement definitions for checking
         membership in a generic defined set";

      uses match-interface-condition;
      uses prefix-set-condition;
      uses neighbor-set-condition;
      uses tag-set-condition;

    }

    grouping policy-conditions {
      description
        "Data for general policy conditions, i.e., those
         not related to match-sets";

        leaf call-policy {
          type leafref {
            path "../../../../../../" +
              "rt-pol:policy-definitions/" +
              "rt-pol:policy-definition/rt-pol:name";
            require-instance true;
          }
          description
            "Applies the statements from the specified policy
             definition and then returns control the current
             policy statement. Note that the called policy may
             itself call other policies (subject to
             implementation limitations). This is intended to
             provide a policy 'subroutine' capability.  The
             called policy should contain an explicit or a
             default route disposition that returns an
             effective true (accept-route) or false
             (reject-route), otherwise the behavior may be
             ambiguous and implementation dependent";
        }

        leaf source-protocol {
          type identityref {
            base rt:control-plane-protocol;
          }
          description
            "Condition to check the protocol / method used to install
```

```
            the route into the local routing table";
        }
    }

    grouping policy-conditions-top {
      description
        "Top-level grouping for policy conditions";

      container conditions {
        description
          "Condition statements for the current policy statement";

        uses policy-conditions;

        uses generic-conditions;
      }
    }

    grouping policy-statements {
      description
        "Data for policy statements";

      leaf name {
        type string;
        description
          "Name of the policy statement";
      }
    }


    grouping policy-actions {
      description
        "Top-level grouping for policy actions";

      container actions {
        description
          "Top-level container for policy action statements";

        leaf policy-result {
          type policy-result-type;
          description
            "Select the final disposition for the route, either
             accept or reject.";
        }
        leaf set-metric {
          type uint32;
          description
            "Set a new metric for the route.";
```

```
        }
        leaf set-preference {
          type uint16;
          description
            "Set a new preference for the route.";
        }
      }
    }

    grouping policy-statements-top {
      description
        "Top-level grouping for the policy statements list";

      container statements {
        description
          "Enclosing container for policy statements";

        list statement {
          key "name";
          ordered-by user;
          description
            "Policy statements group conditions and actions
             within a policy definition.  They are evaluated in
             the order specified (see the description of policy
             evaluation at the top of this module.";

          uses policy-statements;

          uses policy-conditions-top;
          uses policy-actions;
        }
      }
    }


    grouping policy-definitions {
      description
        "This grouping provides policy definitions";

      leaf name {
        type string;
        description
          "Name of the top-level policy definition -- this name
          is used in references to the current policy";
      }
    }

    grouping apply-policy-import {
```

```
      description
        "Grouping for applying import policies";

      leaf-list import-policy {
        type leafref {
          path "/rt-pol:routing-policy/rt-pol:policy-definitions/" +
            "rt-pol:policy-definition/rt-pol:name";
          require-instance true;
        }
        ordered-by user;
        description
          "List of policy names in sequence to be applied on
           receiving a routing update in the current context, e.g.,
           for the current peer group, neighbor, address family,
           etc.";
      }

      leaf default-import-policy {
        type default-policy-type;
        default reject-route;
        description
          "Explicitly set a default policy if no policy definition
           in the import policy chain is satisfied.";
      }

    }

    grouping apply-policy-export {
      description
        "Grouping for applying export policies";

      leaf-list export-policy {
        type leafref {
          path "/rt-pol:routing-policy/rt-pol:policy-definitions/" +
            "rt-pol:policy-definition/rt-pol:name";
          require-instance true;
        }
        ordered-by user;
        description
          "List of policy names in sequence to be applied on
           sending a routing update in the current context, e.g.,
           for the current peer group, neighbor, address family,
           etc.";
      }

      leaf default-export-policy {
        type default-policy-type;
        default reject-route;
```

```
        description
          "Explicitly set a default policy if no policy definition
           in the export policy chain is satisfied.";
      }
    }


    grouping apply-policy {
      description
        "Configuration data for routing policies";

      uses apply-policy-import;
      uses apply-policy-export;

      container apply-policy-state {
        description
          "Operational state associated with routing policy";

        //TODO: identify additional state data beyond the intended
        //policy configuration.
      }

    }


    grouping apply-policy-group {
      description
        "Top level container for routing policy applications. This
         grouping is intended to be used in routing models where
         needed.";

      container apply-policy {
        description
          "Anchor point for routing policies in the model.
           Import and export policies are with respect to the local
           routing table, i.e., export (send) and import (receive),
           depending on the context.";

        uses apply-policy;

      }
    }

    container routing-policy {
      description
        "Top-level container for all routing policy";

      container defined-sets {
```

```
        description
          "Predefined sets of attributes used in policy match
           statements";

        uses prefix-set-top;
        uses neighbor-set-top;
        uses tag-set-top;
      }

      container policy-definitions {
        description
          "Enclosing container for the list of top-level policy
           definitions";

        list policy-definition {
          key "name";
          description
            "List of top-level policy definitions, keyed by unique
             name.  These policy definitions are expected to be
             referenced (by name) in policy chains specified in import
             or export configuration statements.";

          uses policy-definitions;

          uses policy-statements-top;
        }
      }
    }
  }
  <CODE ENDS>
```

11. Policy examples

Below we show an example of XML-encoded configuration data using the routing policy and BGP models to illustrate both how policies are defined, and also how they can be applied. Note that the XML has been simplified for readability.

```
<?yfile include="file:///tmp/routing-policy-example-draft.xml"?>
```

12. References

12.1. Normative references

[I-D.ietf-netmod-intf-ext-yang]
Wilton, R., Ball, D., tsingh@juniper.net, t., and S. Sivaraj, "Common Interface Extension YANG Data Models", draft-ietf-netmod-intf-ext-yang-07 (work in progress), March 2019.

[I-D.ietf-netmod-sub-intf-vlan-model]
Wilton, R., Ball, D., tapsingh@cisco.com, t., and S. Sivaraj, "Sub-interface VLAN YANG Data Models", draft-ietf-netmod-sub-intf-vlan-model-05 (work in progress), March 2019.

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

[RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <https://www.rfc-editor.org/info/rfc3688>.

[RFC4271] Rekhter, Y., Ed., Li, T., Ed., and S. Hares, Ed., "A Border Gateway Protocol 4 (BGP-4)", RFC 4271, DOI 10.17487/RFC4271, January 2006, <https://www.rfc-editor.org/info/rfc4271>.

[RFC6020] Bjorklund, M., Ed., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, DOI 10.17487/RFC6020, October 2010, <https://www.rfc-editor.org/info/rfc6020>.

[RFC6991] Schoenwaelder, J., Ed., "Common YANG Data Types", RFC 6991, DOI 10.17487/RFC6991, July 2013, <https://www.rfc-editor.org/info/rfc6991>.

[RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <https://www.rfc-editor.org/info/rfc7950>.

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

[RFC8340] Bjorklund, M. and L. Berger, Ed., "YANG Tree Diagrams", BCP 215, RFC 8340, DOI 10.17487/RFC8340, March 2018, <https://www.rfc-editor.org/info/rfc8340>.

[RFC8342] Bjorklund, M., Schoenwaelder, J., Shafer, P., Watsen, K., and R. Wilton, "Network Management Datastore Architecture (NMDA)", RFC 8342, DOI 10.17487/RFC8342, March 2018, <https://www.rfc-editor.org/info/rfc8342>.

[RFC8343] Bjorklund, M., "A YANG Data Model for Interface Management", RFC 8343, DOI 10.17487/RFC8343, March 2018, <https://www.rfc-editor.org/info/rfc8343>.

[RFC8349] Lhotka, L., Lindem, A., and Y. Qu, "A YANG Data Model for Routing Management (NMDA Version)", RFC 8349, DOI 10.17487/RFC8349, March 2018, <https://www.rfc-editor.org/info/rfc8349>.

12.2. Informative references

[I-D.ietf-idr-bgp-model] Jethanandani, M., Patel, K., and S. Hares, "BGP YANG Model for Service Provider Networks", draft-ietf-idr-bgp-model-06 (work in progress), June 2019.

Appendix A. Acknowledgements

The routing policy module defined in this draft is based on the OpenConfig route policy model. The authors would like to thank to OpenConfig for their contributions, especially Anees Shaikh, Rob Shakir, Kevin D'Souza, and Chris Chase.

The authors are grateful for valuable contributions to this document and the associated models from: Ebben Aires, Luyuan Fang, Josh George, Stephane Litkowski, Ina Minei, Carl Moberg, Eric Osborne, Steve Padgett, Juergen Schoenwaelder, Jim Uttaro, Russ White, and John Heasley.

Authors' Addresses

Yingzhen Qu
Futurewei
2330 Central Expressway
Santa Clara CA 95050
USA

Email: yingzhen.qu@futurewei.com

Jeff Tantsura
Apstra

Email: jefftant.ietf@gmail.com

Acee Lindem
Cisco
301 Mindenhall Way
Cary, NC 27513
US

Email: acee@cisco.com

Xufeng Liu
Volta Networks

Email: xufeng.liu.ietf@gmail.com

INTERNET-DRAFT
Intended status: Proposed Standard

Z. Li
S. Zhuang
G. Yan
D. Eastlake
Huawei

Expires: May 7, 2019
November 8, 2018

YANG Data Model for Point-to-Point Tunnel Policy
draft-li-rtgwg-tunnel-policy-yang-02

Abstract

This document defines a YANG data model that can be used to configure and manage point-to-point tunnel policy.

Status of this Memo

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

Distribution of this document is unlimited. Comments should be sent to the authors or the TRILL working group mailing list: trill@ietf.org.

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/1id-abstracts.html. The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html.

Table of Contents

1. Introduction

YANG [RFC6020] is a data definition language used to define the contents of a conceptual data store that allows networked devices to be managed using NETCONF [RFC6241]. YANG is proving relevant beyond its initial confines, as bindings to other interfaces (e.g. ReST) and encoding other than XML (e.g. JSON) are being defined. Furthermore, YANG data models can be used as the basis of implementation for other interfaces, such as CLI and programmatic APIs.

This document defines a YANG data model that can be used to configure and manage point-to-point tunnel policy.

2. Definitions and Acronyms

JSON: JavaScript Object Notation

LSP: Label Switched Path

NETCONF: Network Configuration Protocol

RD: Route Distinguisher

TNLM: Tunnel Management

VPN: Virtual Private Network

YANG: A data definition language specified in [RFC6020] for use with NETCONF [RFC6241]

3. Introduction

3.1 Tunnel Policy

Multiple types of tunnels can be used for VPN services, such as LDP LSPs, static LSPs, and CRLSP. It is necessary to select different tunnels for the VPN services to satisfy the required specific tunnel policy.

A tunnel policy determines which type of tunnels can be selected. Tunnel policies can be classified into two modes:

- Selection Sequence: The system selects a tunnel for the service based on the tunnel type priorities defined in the tunnel policy.
- Tunnel binding: The system selects only a specified tunnel for the service.

3.1.1 Selection Sequence

Selection sequence, as a tunnel policy mode, specifies the tunnel-selecting sequence and the number of tunnels in the load balancing mode. Selection Sequence is applicable to the tunnels including the LSP, CR-LSP, etc. In selection-sequence mode, tunnels are selected in sequence. If a tunnel listed earlier is Up and not bound, it is selected regardless of whether other services have selected it; if a tunnel is listed later, it is not selected except when load balancing is required or the preceding tunnels are all in the Down state.

3.1.2 Tunnel Binding

Tunnel binding, as a tunnel policy mode, binds a tunnel with a destination IP address. Tunnel binding is only applicable to TE tunnels.

In tunnel binding mode, multiple TE tunnels can be specified to perform load balancing for the same destination IP address. Moreover, the down-switch attribute can be specified to ensure that other tunnels can be selected when all the designated tunnels are unavailable, which keeps the traffic uninterrupted to the maximum extent.

In terms of tunnel selection among TE tunnels, tunnels are selected according to the destination IP address and name of these TE tunnels.

The principles of tunnel selection are as follows:

1. If the tunnel policy designates no TE tunnel for the destination IP address, the tunnels selection sequence is LSP, CR-LSP.

2. If the tunnel policy designates a TE tunnel for the destination IP address, and the designated TE tunnels is available, that TE tunnel is selected.

3. If the tunnel policy designates a TE tunnel for the destination IP address, but the designated TE tunnels is unavailable, the tunnel-selecting result is determined by the down-switch attribute. If the down-switch attribute is configured, another available tunnel is selected based on the sequence of LSP, CR-LSP, and GRE tunnel; if the down-switch attribute is not configured, no tunnel is selected.

3.2 Tunnel Selector for Routes

A tunnel policy selector defines certain matching rules and associates the routes whose attributes matching the rules with specific tunnels. This facilitates flexible tunneling and better satisfies user requirements.

A tunnel policy selector consists of one more policy nodes and the relationship between these policy nodes is "OR". The system checks the policy nodes based on index numbers. If a route matches a policy node in the tunnel policy, the route does not continue to match the next policy node. Each policy node comprises a set of if-match and apply clauses:

1. The if-match clauses define the matching rules that are used to match certain route attributes such as the next hop and RD. The relationship between the if-match clauses of a node is "AND". A route matches a node only when the route meets all the matching rules specified by the if-match clauses of the node.

2. The apply clause specifies actions. When a route matches a node, the apply clause selects a tunnel policy for the route. The matching modes of a node are as follows:

   a) Permit: If a route matches all the if-match clauses of a node, the route matches the node and the actions defined by the apply clause are performed on the route. If a route does not match one if-match clause of a node, the route continues to match the next node.

   b) Deny: In this mode, the actions defined by the apply clause

are not performed. If a route matches all the if-match clauses of a node, the route is denied and does not match the next node.

3.3 Tunnel Selector for VPNs

Selection of the tunnel for the VPN services includes the matching rules and the applied tunnel policy. The data model is defined in the drafts of VPN Yang models which are out of the scope of this document. They can refer to the Yang models defined in the document for tunnel policy.

4. Design of Data Model

4.1 Tunnel Policy YANG Model

A tunnel policy determines which type of tunnels can be selected by an application module. The configuration of tunnel policy includes defining the tunnel selection sequence mode and the binding mode for the tunnel selection. The nonexistentCheckFlag controls whether the system allows a nonexistent tunnel policy to be specified in a command.

```
+--rw tnlmGlobal
|  +--rw nonexistentCheckFlag?   boolean
+--rw tunnelPolicys
|  +--rw tunnelPolicy* [tnlPolicyName]
|     +--rw tnlPolicyName     string
|     +--ro tnlPolicyExist?   tnlPolicyExist
|     +--ro tpSubCount?       uint32
|     +--rw description?      string
|     +--rw tnlPolicyType?    tnlmbaseTnlPolicyType
|     +--rw tpNexthops
|     |  +--rw tpNexthop* [nexthopIPaddr]
|     |     +--rw nexthopIPaddr      inet:ipv4-address-no-zone
|     |     +--rw downSwitch?        boolean
|     |     +--rw ignoreDestCheck?   boolean
|     |     +--rw isIncludeLdp?      boolean
|     |     +--rw tpTunnels
|     |        +--rw tpTunnel* [tunnelName]
|     |           +--rw tunnelName    string
|     +--rw tnlSelSeqs
|        +--rw tnlSelSeq!
|           +--rw loadBalanceNum?   uint32
|           +--rw selTnlType1?      tnlmbaseSelTnlType
|           +--rw selTnlType2?      tnlmbaseSelTnlType
|           +--rw selTnlType3?      tnlmbaseSelTnlType
|           +--rw selTnlType4?      tnlmbaseSelTnlType
|           +--rw selTnlType5?      tnlmbaseSelTnlType
|           +--rw selTnlType6?      tnlmbaseSelTnlType
|           +--rw unmix?            boolean
```

4.2 Tunnel Selector YANG Model

A tunnel policy selector defines certain matching rules and associates the routes whose attributes matching the rules with specific tunnels. This facilitates flexible tunneling satisfying user requirements.

Configuration of the tunnel selector and applying it to the BGP VPNv4/VPNv6 address-family can make the VPN service select the specific tunnel for VPN data transmission.

```
+--rw tunnelSelectors
+--rw tunnelSelector* [name]
   +--rw name                    string
   +--rw tunnelSelectorNodes
      +--rw tunnelSelectorNode* [nodeSequence]
         +--rw nodeSequence      uint32
         +--rw matchMode         rtpMatchMode
         +--rw matchCondition
         |  +--rw matchDestPrefixFilters
         |  |  +--rw matchDestPrefixFilter!
         |  |     +--rw prefixName?   string
         |  +--rw matchIPv4NextHops
         |  |  +--rw matchIPv4NextHop!
         |  |     +--rw matchType?      rtpTnlSelMchType
         |  |     +--rw prefixName?     string
         |  |     +--rw aclNameOrNum?   string
         |  +--rw matchIPv6NextHops
         |  |  +--rw matchIPv6NextHop!
         |  |     +--rw ipv6PrefixName?   string
         |  +--rw matchCommunityFilters
         |  |  +--rw matchCommunityFilter* [cmntyNameOrNum]
         |  |     +--rw cmntyNameOrNum    string
         |  |     +--rw wholeMatch?       boolean
         |  |     +--rw sortMatch?        boolean
         |  +--rw matchRdFilters
         |     +--rw matchRdFilter!
         |        +--rw rdIndex?   uint32
         +--rw applyAction
            +--rw applyTnlPolicys
               +--rw applyTnlPolicy!
                  +--rw tnlPolicyName?   string

augment /bgp:bgp/bgp:global/bgp:afi-safis/bgp:afi-safi/
                                       bgp:l3vpn-ipv4-unicast:
   +--rw tunnelSelectorName?   string
augment /bgp:bgp/bgp:global/bgp:afi-safis/bgp:afi-safi/
                                       bgp:l3vpn-ipv6-unicast:
   +--rw tunnelSelectorName?   string
```

5. Tunnel Policy Yang Module

```
//Tunnel Policy YANG MODEL
<CODE BEGINS> file " tunnel-policy@2018-09-15.yang "
module tunnel-policy {
  namespace "urn:huawei:params:xml:ns:yang:tunnel-policy";
  // replace with IANA namespace when assigned
  prefix tnlp;

  import ietf-bgp {
 prefix bgp;

  }

  import ietf-inet-types {
 prefix inet;
 //rfc6991-Common YANG Data Types
  }

  organization
 "Huawei Technologies Co., Ltd.";
  contact
 "Huawei Industrial Base Bantian, Longgang Shenzhen 518129
     People's Republic of China
     Website: http://www.huawei.com Email: support@huawei.com";
  description
 "This YANG module defines the tunnel policy configuration
  data for tunnel policy service.

  VPN data needs to be carried by tunnels. By default, the
  system selects LSPs to carry VPN services without performing
  load balancing. If this cannot meet the requirements of VPN
  services, a tunnel policy needs to be used. The tunnel policy
  may be a tunnel type prioritizing policy or a tunnel binding
  policy. Determine which type of tunnel policy to use based
  on your actual requirements:
  * A tunnel type prioritizing policy can change the tunnel
    type selected for VPN services and allow load balancing
    among tunnels.
  * A tunnel binding policy can bind a VPN service to
    specified MPLS TE tunnels to provide QoS guarantee for
    the VPN service.

  Terms and Acronyms

  ... ";

  revision 2018-09-15 {
 description
   "Initial revision.";
```

```
     }

     typedef tnlmbaseTnlPolicyType {
    type enumeration {
      enum "invalid" {
        description
          "Tunnel policy with null configurations.";
      }
      enum "tnlSelectSeq" {
        description
          "Tunnel select-seq policy. This policy allows you
           to specify the sequence in which different types
           of tunnels are selected and the number of tunnels
           for load balancing.";
      }
      enum "tnlBinding" {
        description
          "Tunnel binding policy. This policy allows you to
           specify the next hop to be bound to a TE tunnel.
           After a TE tunnel is bound to a destination
           address, VPN traffic destined for that destination
           address  will be transmitted over the TE tunnel.";
      }
    }
    description
      "tunnel policy type";
     }
     typedef tnlmbaseSelTnlType {
    type enumeration {
      enum "invaild" {
        description
          "Search for invalid tunnels.";
      }
      enum "lsp" {
        description
          "Search for LDP LSPs.";
      }
      enum "cr-lsp" {
        description
          "Search for CR-LSPs.";
      }
      enum "gre" {
        description
          "Search for GREs.";
      }
      enum "ldp" {
        description
          "Search for LDP LSPs.";
      }
      enum "bgp" {
```

```
        description
          "Search for BGP LSPs.";
      }
      enum "srbe-lsp" {
        description
          "Search for SR-LSPs.";
      }
      enum "sr-te" {
        description
          "Search for SR-TE.";
      }
      enum "te" {
        description
          "Search for TE.";
      }
    }
    description
      "tunnel select type";
     }

     typedef tnlPolicyExist {
    type enumeration {
      enum "true" {
        description
          "The tunnel policy has been configured.";
      }
      enum "false" {
        description
          "The tunnel policy has not been configured.";
      }
    }
    description
      "tunnel policy state";
     }

     typedef rtpMatchMode {
    type enumeration {
      enum "permit" {
        description
          "Matching mode of filters.";
      }
      enum "deny" {
        description
          "Matching mode of filters.";
      }
    }
    description
      "match mode";
     }
```

```
     typedef rtpTnlSelMchType {
    type enumeration {
      enum "matchNHopPF" {
        description
          "Match IPv4 next hops by an IPv4 prefix.";
      }
      enum "matchNHopAcl" {
        description
          "Match IPv4 next hops by an ACL.";
      }
    }
    description
      "tunnel selector type";
     }

     /*
   A tunnel policy determines which type of tunnels can be
   selected by an application module.

   Tunnel policies can be classified into two modes:
   Select-seq: The system selects a tunnel for an application
   program based on the tunnel type priorities defined in the
   tunnel policy.
   Tunnel binding: The system selects only a specified tunnel
   for an application program.

   The two modes are mutually exclusive.

   Configuration example:
   #
   tunnel-policy policy1
    description policy1
    tunnel binding destination 1.1.1.1 te Tunnel0/0/0 down-switch
   #
   tunnel-policy policy2
    tunnel select-seq cr-lsp gre lsp load-balance-number 2
   #
   tunnel-policy policy3
    tunnel binding destination 1.1.1.1 te Tunnel0/0/0 down-switch
    tunnel binding destination 3.3.3.3 te Tunnel0/0/0
                                           ignore-destination-check
    tunnel binding destination 5.5.5.5 te Tunnel0/0/0
   #
     */

     container tnlmGlobal {
    description
      "Global parameters for tunnel policy.";
    leaf nonexistentCheckFlag {
      type boolean;
```

```
      default "true";
      description
        "Nonexistent config check flag of tunnel policy.
         By default, if you specify a nonexistent tunnel policy
         in a command, the command does not take effect. To enable
         the system to allow a nonexistent tunnel policy to be
         specified in a command, run the tunnel-policy
         nonexistent-config-check disable command.";
    }
     }

     container tunnelPolicys {
    description
      "List of global tunnel policy configurations. A tunnel
       policy can  be used to specify a rule for selecting
       tunnels.";

    list tunnelPolicy {
      key "tnlPolicyName";

      description
        "A policy for selecting tunnels to carry services. The
         tunnel management module searches for and returns the
         required tunnels based on the tunnel policy. By default,
         no tunnel policy is configured, the system selects an
         available tunnel in the order of conventional LSPs,
         CR-LSPs, and Local_IFNET LSPs, and load balancing is
         not performed.";

      leaf tnlPolicyName {
        type string {
          length "1..39";
        }
        description
          "Name of a tunnel policy. The value is a string of 1 to
           39 case-sensitive characters, spaces not supported.";
      }
      leaf tnlPolicyExist {
        type tnlPolicyExist;
        config false;
        description
          "Whether a tunnel policy has been configured.";
      }
      leaf tpSubCount {
        type uint32;
        config false;
        description
          "Number of times a tunnel policy is referenced.";
      }
      leaf description {
```

```
        type string {
          length "1..80";
        }
        description
          "Description of a tunnel policy.";
      }

      leaf tnlPolicyType {
        type tnlmbaseTnlPolicyType;
        default "invalid";
        description
          "Tunnel policy type. The available options are sel-seq,
           binding, and invalid. A tunnel policy can be configured
           with only one policy type.";
      }
      container tpNexthops {
        must "not(../tnlPolicyType='tnlBinding') or "
             + "(../tnlPolicyType='tnlBinding' "
             + "and count(tpNexthop)>=1)";
        description
          "List of tunnel binding configurations.";
        list tpNexthop {
          when "not(../../tnlPolicyType='tnlSelectSeq') or "
               + "../../tnlPolicyType='tnlBinding'";
          key "nexthopIPaddr";
          max-elements  "65535";
          description
            "Rule for binding a TE tunnel to a destination address,
             so that the VPN traffic destined for that destination
             address can be transmitted over the TE tunnel.";
          leaf nexthopIPaddr {
            type inet:ipv4-address-no-zone;
            description
              "Destination IP address to be bound to a tunnel.";
          }
          leaf downSwitch {
            type boolean;
            default "false";
            description
              "Enable tunnel switching. After this option is
               selected, if the bound TE tunnel is unavailable,
               the system will select an available tunnel in
               the order of conventional LSPs, CR-LSPs, and
               Local_IFNET tunnels.";
          }
          leaf ignoreDestCheck {
            type boolean;
            default "false";
            description
              "Do not check whether the destination address of the
```

```
            TE tunnel matches the destination address specified
            in the tunnel policy.";
        }
        leaf isIncludeLdp {
          type boolean;
          must "(../isIncludeLdp='true' and not "
               + "(../downSwitch='true')) or "
               + "../isIncludeLdp='false'";
          default "false";
          description
            "Is loadbalance with LDP";
        }
        container tpTunnels {
          description
            "List of tunnels available for an application.";
          list tpTunnel {
            key "tunnelName";
            min-elements  "1";
            max-elements  "16";
            description
              "Tunnel.";
            leaf tunnelName {
              type string {
                length "1..47";
              }
              description
                "Name of the specified tunnel.";
            }
          }
        }
      }
    }
    container tnlSelSeqs {
      when "not(../tnlPolicyType='invalid' or "
           + "../tnlPolicyType='tnlBinding')";
      must "not(../tnlPolicyType='tnlSelectSeq') or "
           + "(../tnlPolicyType='tnlSelectSeq' and "
           + "count(tnlSelSeq)>=1)";
      description
        "Sequence in which different types of tunnels are
         selected.
         If the value is INVALID, no tunnel type has been
         configured.";
      container tnlSelSeq {
        when "not(../../tnlPolicyType='invalid' or "
             + "../../tnlPolicyType='tnlBinding') or "
             + "../../tnlPolicyType='tnlSelectSeq'";
        presence "create tnlSelSeq";
        description
          "Sequence in which different types of tunnels are
```

```
           selected. If the value is INVALID, no tunnel type
           has been configured.";
        leaf loadBalanceNum {
          type uint32 {
            range "1..64";
          }
          default "1";
          description
            "Sequence in which different types of tunnels are
             selected. The available tunnel types are CR-LSP,
             and LSP. LSP tunnels refer to LDP LSP tunnels
             here.";
        }
        leaf selTnlType1 {
          type tnlmbaseSelTnlType;
          default "invaild";
          description
            "Sequence in which different types of tunnels are
             selected. If the value is INVALID, no tunnel type
             has been configured.";
        }
        leaf selTnlType2 {
          when "not(../selTnlType1='invaild' and "
               + "../../../tnlPolicyType='tnlSelectSeq' or "
               + "../selTnlType1='invaild')";
          type tnlmbaseSelTnlType;
          default "invaild";
          description
            "Sequence in which different types of tunnels are
             selected. If the value is INVALID, no tunnel type
             has been configured.";
        }
        leaf selTnlType3 {
          when "not(../selTnlType1='invaild' or "
               + "../selTnlType2='invaild')";
          type tnlmbaseSelTnlType;
          default "invaild";
          description
            "Sequence in which different types of tunnels are
             selected. If the value is INVALID, no tunnel type
             has been configured.";
        }
        leaf selTnlType4 {
          when "not(../selTnlType1='invaild' or "
               + "../selTnlType2='invaild' or "
               + "../selTnlType3='invaild')";
          type tnlmbaseSelTnlType;
          default "invaild";
          description
            "Sequence in which different types of tunnels are
```

```
              selected. If the value is INVALID, no tunnel type
              has been configured.";
          }
          leaf selTnlType5 {
            when "not(../selTnlType1='invaild' or "
                 + "../selTnlType2='invaild' or "
                 + "../selTnlType3='invaild' or "
                 + "../selTnlType4='invaild')";
            type tnlmbaseSelTnlType;
            default "invaild";
            description
              "Sequence in which different types of tunnels are
               selected. If the value is INVALID, no tunnel type
               has been configured.";
          }
          leaf selTnlType6 {
            when "not(../selTnlType1='invaild' or "
                 + "../selTnlType2='invaild' or "
                 + "../selTnlType3='invaild' or "
                 + "../selTnlType4='invaild' or "
                 + "../selTnlType5='invaild')";
            type tnlmbaseSelTnlType;
            default "invaild";
            description
              "Sequence in which different types of tunnels are
               selected. If the value is INVALID, no tunnel type
               has been configured.";
          }
          leaf unmix {
            type boolean;
            default "false";
            description
              "unmix flag.";
          }
        }
      }
    }

     }//End of container tunnelPolicys

     /*
     The tunnel selector is specific to BGP/MPLS IP VPN services
     (a type of VPN service), selecting a tunnel policy for
     VPNv4/VPNv6 routes on the backbone network.

     A tunnel selector selects tunnel policies for routes after
     filtering routes based on some route attributes such as the
     route distinguisher (RD) and next hop. This makes tunnel
     selection more flexible.
```

```
     A tunnel selector is often used on the autonomous system
     boundary router (ASBR) in inter-AS VPN Option B or the
     superstratum provider edge (SPE) in hierarchy of VPN (HoVPN).
     */
     container tunnelSelectors {
    description
      "List of tunnel selectors.";
    list tunnelSelector {
      key "name";
      max-elements  "65535";
      description
        "Tunnel selector. Usually used in BGP VPN Option B or
         BGP VPN Option C, tunnel selector selects a proper
         tunnel policy for routes.";

      leaf name {
        type string {
          length "1..40";
        }
        description
          "Name of a tunnel selector. The name is a string of
           1 to 40 case-sensitive characters without spaces.";
      }

      container tunnelSelectorNodes {
        description
          "List of tunnel selector nodes.";
        list tunnelSelectorNode {
          key "nodeSequence";
          min-elements  "1";
          max-elements  "65535";

          leaf nodeSequence {
            type uint32 {
              range "0..65535";
            }
            description
               "Sequence number of a node.
                Specifies the index of a node of the tunnel
                selector.
                When a route-policy is used to filter a route,
                the route first matches the node with the
                smallest node value.";
          }
          leaf matchMode {
            type rtpMatchMode;
            mandatory true;
            description
              "Matching mode of nodes.";
          }
```

```
          container matchCondition {
            description
              "Match Type List";

            container matchDestPrefixFilters {
              description
                "Match IPv4 destination addresses by the prefix
                 filter. The configurations of matching IPv4
                 destination addresses by the prefix filter are
                 mutually exclusive with the configurations of
                 matching IPv4 destination addresses based on
                 ACL rules.";

              container matchDestPrefixFilter {
                presence "create matchDestPrefixFilter";
                description
                  "Match an IPv4 destination address by the prefix
                   filter.";
                leaf prefixName {
                  type "string";
                  description
                    "Name of the specified prefix filter when IPv4
                     destination addresses are matched.";
                }
              }
            } // End of matchDestPrefixFilters

            container matchIPv4NextHops {
              description
                "Match IPv4 next hops by the prefix filter or ACL
                 filter. The configurations of matching IPv4 next
                 hops by the prefix filter are mutually exclusive
                 with the configurations of matching IPv4 next
                 hops by the ACL filter.";

              container matchIPv4NextHop {
                presence "create matchIPv4NextHop";
                description
                  "Match an IPv4 next hop by the prefix or ACL.";
                leaf matchType {
                  type rtpTnlSelMchType;
                  description
                    "Match type. IPv4 next hops are matched with
                     either the prefix or ACL.";
                }
                leaf prefixName {
                  when "not(../matchType='matchNHopAcl' or "
                       + "not(../matchType)) or "
                       + "../matchType='matchNHopPF'";
                  type "string";
```

```
            description
              "Name of the specified prefix when IPv4 next hops
               are matched.";
          }
          leaf aclNameOrNum {
            when "not(../matchType='matchNHopPF' or "
                 + "not(../matchType)) or "
                 + "../matchType='matchNHopAcl'";
            type string {
              length "1..32";
            }
            description
              "Name of the specified ACL when next hops are
               matched, which can be a value ranging from
               2000 to 2999 or a string beginning with a-z
               or A-Z.";
          }
        }
      } //End of container matchIPv4NextHops

      container matchIPv6NextHops {
        description
          "Match IPv6 next hops by the IPv6 prefix filter.";
        container matchIPv6NextHop {
          presence "create matchIPv6NextHop";
          description
            "Match an IPv6 next hop by the IPv6 prefix
             filter.";

          leaf ipv6PrefixName {
            type "string";
            description
              "Name of the specified prefix filter when IPv6
               next hops are matched.";
          }
        }
      } //End of container matchIPv6NextHops

      container matchCommunityFilters {
        description
          "Match community attribute filters.";
        list matchCommunityFilter {
          key "cmntyNameOrNum";
          max-elements  "32";
          description
            "Match a community attribute filter.";
          leaf cmntyNameOrNum {
            type string {
              length "1..51";
              pattern '((0*[1-9][0-9]?)|(0*1[0-9][0-9])|'
```

```
                    + '([^?-9][^?0,50})|'
                    + '([][^?^?-9][^?)';
                }
                description
                  "Name or index of a community attribute filter.
                   It can be a numeral or a string. The ID of a
                   basic community attribute filter is an integer
                   ranging from 1 to 99; the ID of an advanced
                   community attribute filter is an integer
                   ranging from 100 to 199. The name of a community
                   attribute filter is a string of 1 to 51
                   characters. The string cannot contain only
                   digits.";
              }
              leaf wholeMatch {
                type boolean;
                default "false";
                description
                  "All the communities are matched. It is valid to
                   only basic community attribute filters.";
              }
              leaf sortMatch {
                type boolean;
                default "false";
                description
                  "Match all community attributes in sequence.It
                   is valid to only Advanced community attribute
                   filters.";
              }
            }
          } //End of container matchCommunityFilters

          container matchRdFilters {
            description
              "Match RD filters.";
            container matchRdFilter {
              presence "create matchRdFilter";
              description
                "Match an RD filter.";
              leaf rdIndex {
                type uint32 {
                  range "1..1024";
                }
                description
                  "Index of an RD filter.";
              }
            }
          } //End of container matchRdFilters

        }//End of container matchCondition
```

```
          container applyAction {
            description
              "Set Type List";
            container applyTnlPolicys {
              description
                "Set tunnel policies.";
              container applyTnlPolicy {
                presence "create applyTnlPolicy";
                description
                  "Set a tunnel policy.";
                leaf tnlPolicyName {
                  type string {
                    length "1..39";
                  }
                  description
                    "Name of a tunnel policy. The name is a
                     string of 1 to 39 case-sensitive characters,
                     spaces not supported.";
                }
              }
            }
          } //End of container applyAction

        }

      } //End of container tunnelSelectorNodes

    } //End of list tunnelSelector

     } //End of container tunnelSelectors

     /*
   * augment some bgp vpn functions in bgp module.
   */
     augment "/bgp:bgp/bgp:global/bgp:afi-safis/" +.............
               "bgp:afi-safi/bgp:l3vpn-ipv4-unicast" {
    leaf tunnelSelectorName {
      description
        "Specifies the name of a tunnel selector.";

      type "string";
    }
     }

     augment "/bgp:bgp/bgp:global/bgp:afi-safis/" +.............
               "bgp:afi-safi/bgp:l3vpn-ipv6-unicast" {
    leaf tunnelSelectorName {
      description
        "Specifies the name of a tunnel selector.";
```

```
      type "string";
    }
     }
   <CODE ENDS>
```

6. IANA Considerations

This document requires no IANA actions.

7. Security Considerations

The YANG module specified in this document defines a schema for data that is designed to be accessed via network management protocols such as NETCONF [RFC6241] or RESTCONF [RFC8040]. The lowest NETCONF layer is the secure transport layer, and the mandatory-to-implement secure transport is Secure Shell (SSH) [RFC6242]. The lowest RESTCONF layer is HTTPS, and the mandatory-to-implement secure transport is TLS [RFC8446].

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

There are a number of data nodes defined in this YANG module that are writable/creatable/deletable (i.e., config true, which is the default). These data nodes may be considered sensitive or vulnerable in some network environments. Write operations (e.g., edit-config) to these data nodes without proper protection can have a negative effect on network operations. These are the subtrees and data nodes and their sensitivity/vulnerability:

tbd

Unauthorized access to any data node of these subtrees can adversely affect ... tbd ...

Some of the readable data nodes in this YANG module may be considered sensitive or vulnerable in some network environments. It is thus important to control read access (e.g., via get, get-config, or notification) to these data nodes. These are the subtrees and data nodes and their sensitivity/vulnerability:

tbd

Unauthorized access to any data node of these subtrees can disclose ... tbd ...

Acknowledgements

The authors would like to thank the following for their contributions to this work:

Xianping Zhang, Linghai Kong, Xiangfeng Ding, Haibo Wang, and Walker Zheng

Informational References

[RFC6241] Enns, R., Bjorklund, M., Schoenwaelder, J., and A. Bierman, "Network Configuration Protocol (NETCONF)", RFC 6241, June 2011.

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

[RFC8341] Bierman, A. and M. Bjorklund, "Network Configuration Access Control Model", STD 91, RFC 8341, DOI 10.17487/RFC8341, March 2018, <https://www.rfc-editor.org/info/rfc8341>.

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

[RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/info/rfc8446>.

Normative References

[RFC6020] Bjorklund, M., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, October 2010.

Authors' Addresses

Zhenbin Li
Huawei Technologies
Huawei Bld., No.156 Beiqing Rd.
Beijing 100095 China

Email: lizhenbin@huawei.com

Shunwan Zhuang
Huawei Technologies
Huawei Bld., No.156 Beiqing Rd.
Beijing 100095 China

Email: zhuangshunwan@huawei.com

Gang Yan
Huawei Technologies
Huawei Bld., No.156 Beiqing Rd.
Beijing 100095 China

Email: yangang@huawei.com

Donald Eastlake, 3rd
Huawei Technologies
1424 Pro Shop Court
Davenport, FL 33896 USA

Phone: +1-508-333-2270
Email: d3e3e3@gmail.com

Copyright and IPR Provisions

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: June 18, 2016

Z. Li
X. Chen
Huawei Technologies

December 16, 2015

Yang Data Model for Unified Tunnel
draft-li-rtgwg-utunnel-yang-01

Abstract

This document defines a YANG data model for the unified tunnel. The data model includes the operational state of tunnels.

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 June 18, 2016.

Copyright Notice

Table of Contents

1. Introduction

YANG [RFC6020] is a data definition language that was introduced to define the contents of a conceptual data store that allows networked devices to be managed using NETCONF[RFC6241]. YANG is proving relevant beyond its initial confines, as bindings to other interfaces(e.g. ReST) and encoding other than XML (e.g. JSON) are being defined. Furthermore, YANG data models can be used as the basis of implementation for other interface, such as CLI and Programmatic APIs.

At present, multiple types of tunnels can be provided to carry vpn services, such as LDP LSPs, RSVP-TE tunnel, GRE tunnel. All of the tunnels can be managed unifiedly to support vpn services according to tunnel policy. Different types of tunnels can be managed unifiedly to form tunnel-group to support services.

This document defines a YANG data model for the unified tunnel. The data model covers the operational state of tunnels.

2. 2. UTunnel Data Model Organization

2.1. Overview

The information of unified tunnel is about the generic operational state of different types of tunnels. These core data is necessary for vpn service to select the carrying tunnel. The tunnels include p2p, mp2p and p2mp tunnel. P2p and mp2p tunnel use the same container with the name p2ptunnels. P2mp tunnel is not included in this version and will be defined later.

Some tunnels like ldp can be from specific vrf or default vrf. And some tunnels like rsvp-te are only from the default vrf. So the vrf-name is as key attribute of the tunnel.

The tunnel destination can be ipv4 or ipv6 address. It depends on implementation of the tunnel protocol.

The figure below describes the overall structure of the utunnel yang model :

```
module: utunnel
   +--ro utunnel
      +--ro p2ptunnels
         +--ro tunnel* [vrf-name tunnel-protocol tunnel-id]
            +--ro vrf-name           rt:routing-instance-ref
            +--ro tunnel-protocol    tunnel-protocol
            +--ro tunnel-id          tunnel-id
            +--ro tunnel-name?       tunnel-name
            +--ro oper-status?       tunnel-status
            +--ro address-family
               +--ro ipv4
               |  +--ro destination?   inet:ipv4-address
               +--ro ipv6
                  +--ro destination?   inet:ipv6-address
```

3. UTunnel Yang Module

```
<CODE BEGINS> file "ietf-utunnel@2015-12-16.yang"
module ietf-utunnel  {
    namespace "urn:ietf:params:xml:ns:yang:ietf-utunnel";
    // replace with IANA namespace when assigned
    prefix "utunnel";

    import ietf-inet-types {
        prefix "inet";
    }

    import ietf-routing {
        prefix "rt";
    }

    organization "huawei";
    contact
       "lizhenbin@huawei.com
        xiachen@huawei.com
        vinods.kumar@huawei.com";

    description "unified tunnel model";
```

```
    revision "2015-12-16" {
        description "2015-07-05:Initial revision"+
                    "2015-12-16:Add the description in the yang module.";
        reference "TBD";
    }

    typedef tunnel-id {
        type uint32;
        description
            "An identifier for a tunnel.";
    }

    typedef tunnel-name {
        type string {
            length "1..64";
        }
        description
            "The name of a tunnel.";
    }

    typedef tunnel-protocol{
        type enumeration {
            enum ldp {
                value "0";
                description "LDP";
            }
            enum rsvp-te {
                value "1";
                description "RSVP-TE";
            }
            enum bgp {
                value "2";
                description "Labeled BGP";
            }
            enum cr-static-lsp {
                value "3";
                description "Static LSP with TE constraints";
            }
            enum gre {
                value "4";
                description "GRE";
            }
            enum ipsec {
                value "5";
                description "IPSec";
            }
            enum vxlan {
                value "6";
```

```
                description "VXLAN";
            }
            enum segment-routing-best-effort {
                value "7";
                description "Segment Routing BE";
            }
            enum segment-routing-te {
                value "8";
                description "Segment Routing TE";
            }
        }
        description
            "The protocol which is used to set up a tunnel.";
    }

    typedef tunnel-status {
        type enumeration {
            enum up {
                value "0";
                description
                "The status of tunnel is up.";
            }
            enum down {
                value "1";
                description
                "The status of tunnel is down.";
            }
        }
        description
            "The status of a tunnel.";
    }

    container utunnel{
        config false;
        description "The information of unified tunnel. The tunnels include P2P,
 MP2P and P2MP tunnel.
         P2P and MP2P tunnel use the same container with name P2P tunnels. P2MP
tunnel is defined later.";

        container p2ptunnels{
            description "The information of unified P2P tunnel.";

            list tunnel {
                key "vrf-name tunnel-protocol tunnel-id";
                description "List of unified P2P tunnels.";

                leaf vrf-name {
                    type rt:routing-instance-ref;
                    description "The name of vrf. Some tunnels like LDP can be f
rom specific VRF or default VRF.
                                 And some tunnels like RSVP-TE are only from the
 default VRF.";
```

```
                }
                leaf tunnel-protocol {
                    type tunnel-protocol;
                    description "The type of tunnel protocol which can be LDP, G
RE, BGP etc.";
                }
                leaf tunnel-id {
                    type tunnel-id;
                    description "The identifier of tunnel which can be uniquely
represented a tunnel with vrf-name and tunnel-protocol.";
                }
                leaf tunnel-name {
                    type tunnel-name;
                    description "The name of tunnel.";
                }
                leaf oper-status {
                    type tunnel-status;
                    description "The status of tunnel which can be up or down.";
                }
                container address-family {
                    description "IPv4 or IPv6 address family.";

                    container ipv4 {
                        description "IPv4 address family.";
                        leaf destination {
                            type inet:ipv4-address;
                            description "The destination is IPv4 address.";
                        }
                    }  // ipv4
                    container ipv6 {
                        description "IPv6 address family.";
                        leaf destination {
                            type inet:ipv6-address;
                            description "The destination is ipv6 address.";
                        }
                    }  // ipv6
                }
            }
        }
    }
}
<CODE ENDS>
```

4. IANA Considerations

This document registers the following URIs in the IETF XML registry [RFC3688]. Following the format in [RFC3688], the following registration is requested to be made.

URI: urn:ietf:params:xml:ns:yang:ietf-utunnel XML: N/A, the requested URI is an XML namespace.

This document registers a YANG module in the YANG Module Names registry [RFC6020].

name: ietf-utunnel namespace: urn:ietf:params:xml:ns:yang:ietf-utunnel prefix: utunnel.

5. Security Considerations

The YANG module defined in this memo is designed to be accessed via the NETCONF protocol[RFC6241]. The lowest NETCONF layer is the secure transport layer and the mandatory-to-implement secure transport is SSH [RFC6242]. The NETCONF access control model [RFC6536] provides means to restrict access for particular NETCONF users to a pre-configured subset of all available NETCONF protocol operations and content. There are a number of data nodes defined in the YANG module which are writable/creatable/deletable (i.e., config true, which is the default). These data nodes may be considered sensitive or vulnerable in some network environments. Write operations (e.g., <edit-config>) to these data nodes without proper protection can have a negative effect on network operations.

6. Acknowledgements

The authors would like to thank vinod kumar S for his contributions to this work.

7. Normative References

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

[RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <http://www.rfc-editor.org/info/rfc3688>.

[RFC6020] Bjorklund, M., Ed., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, DOI 10.17487/RFC6020, October 2010, <http://www.rfc-editor.org/info/rfc6020>.

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

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

[RFC6536] Bierman, A. and M. Bjorklund, "Network Configuration Protocol (NETCONF) Access Control Model", RFC 6536, DOI 10.17487/RFC6536, March 2012, <http://www.rfc-editor.org/info/rfc6536>.

Authors' Addresses

Zhenbin Li
Huawei Technologies
Huawei Bld., No.156 Beiqing Rd.
Beijing 100095
China

Email: lizhenbin@huawei.com

Xia Chen
Huawei Technologies
Huawei Bld., No.156 Beiqing Rd.
Beijing 100095
China

Email: jescia.chenxia@huawei.com

RTG Working Group
Internet-Draft
Intended status: Standards Track
Expires: June 18, 2016

Q. Liang
G. Yan
S. Zhuang
Huawei Technologies

December 16, 2015

A YANG Data Model for Static Route
draft-liang-rtgwg-staticrt-yang-cfg-01

Abstract

This document defines a YANG data model for static routes. The data model includes configuration data and state data.

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 June 18, 2016.

Copyright Notice

Table of Contents

1. Introduction

Static routes are special routes that are configured manually by network administrators. On simple networks, the network administrator configures static routes in the routers which cannot run dynamic routing protocols or cannot generate routes to a special destination network, so that the network can run properly. Route selection can be controlled using static routes. However, each time a fault occurs on the network or the network topology changes, maybe the network administrator need to reconfigure the static routes.

This document defines a YANG [RFC6020] data model for the configuration and state data of static route. Any RPC or notification definition is not part of this document. . YANG is proving relevant beyond its initial confines, as bindings to other interfaces(e.g. ReST) and encoding other than XML (e.g. JSON) are being defined. Furthermore, YANG data models can be used as the basis of implementation for other interface, such as CLI and Programmatic APIs.

As many vendors have different object constructs to represent the same static route, it has been tried to design this model in a very flexible, extensible and generic way to fit into most of the vendor requirements.

1.1. Terminology

The following terms are defined in [RFC6020] :

- configuration data
- data model
- module
- state data

The terminology for describing YANG data models is found in [RFC6020].

1.2. Tree Diagrams

A simplified graphical representation of the data model is used in this document. The meaning of the symbols in these diagrams is as follows:

- Brackets "[" and "]" enclose list keys.
- Abbreviations before data node names: "rw" means configuration data (read-write), and "ro" means state data (read-only).
- Symbols after data node names: "?" means an optional node, "!" means a presence container, and "*" denotes a list and leaf-list.
- Parentheses enclose choice and case nodes, and case nodes are also marked with a colon (":").
- Ellipsis ("...") stands for contents of subtrees that are not shown.

2. Static Route Data Model

The data model has the following structure for configuration of static routes:

```
module: ietf-staticrt
   +--rw staticrt
   |  +--rw staticrt-cfg
   |  |  +--rw staticrt-topo* [VRF-name address-family topo-name]
   |  |     +--rw topo-name           string
   |  |     +--rw VRF-name            string
   |  |     +--rw address-family      identityref
   |  |     +--rw enable-FRR?         boolean
```

```
|  |     +--rw staticrt-entries* [ip-prefix ip-prefix-mask
|  |           output-interface target-VRF-name target-nexthop]
|  |        +--rw ip-prefix                          inet:ip-address
|  |        +--rw ip-prefix-mask                     inet:ip-address
|  |        +--rw output-interface                   string
|  |        +--rw target-VRF-name                    string
|  |        +--rw target-nexthop                     inet:ip-address
|  |        +--rw description?                       string
|  |        +--rw preference?                        uint32
|  |        +--rw cost?                              uint32
|  |        +--rw tag?                               uint32
|  |        +--rw enable-inherit-cost?               boolean
|  |        +--rw is-permanent?                      boolean
|  |        +--rw not-advertise?                     boolean
|  |        +--rw is-iterated-to-host-route?         boolean
|  |        +--rw BFD-enable?                        boolean
|  |        +--rw BFD-session-name?                  string
|  |        +--rw adminstated-down-not-selected?     boolean
|  +--ro staticrt-state
|     +--ro staticrt-rib
|     |  +--ro staticrt-route*
|     |     +--ro index?                            uint32
|     |     +--ro topo-name?                        string
|     |     +--ro VRF-name?                         string
|     |     +--ro address-family?                   string
|     |     +--ro ip-prefix                         inet:ip-address
|     |     +--ro ip-prefix-mask                    inet:ip-address
|     |     +--ro output-interface?                 string
|     |     +--ro target-VRF-name?                  string
|     |     +--ro target-nexthop?                   inet:ip-address
|     |     +--ro description?                      string
|     |     +--ro preference?                       uint32
|     |     +--ro cost?                             uint32
|     |     +--ro tag?                              uint32
|     |     +--ro enable-inherit-cost?              boolean
|     |     +--ro is-permanent?                     boolean
|     |     +--ro not-advertise?                    boolean
|     |     +--ro is-iterated-to-host-route?        boolean
|     |     +--ro BFD-enable?                       boolean
|     |     +--ro BFD-session-name?                 string
|     |     +--ro adminstated-down-not-selected?    boolean
|     |     +--ro iterated-nexthop?                 inet:ip-address
|     |     +--ro iterated-output-interface?        string
|     |     +--ro label?                            uint32
|     |     +--ro route-state?                      string
|     |     +--ro interface-state?                  string
|     |     +--ro BFD-state?                        string
|     |     +--ro local-address?                    inet:ip-address
```

```
|     |     +--ro remote-address?                  inet:ip-address
|     +--ro staticrt-statistics
|        +--ro staticrt-stats*
|           +--ro topo-name?            string
|           +--ro VRF-name?             string
|           +--ro address-family?       string
|           +--ro static-route-stats*
|              +--ro ip-prefix                     inet:ip-address
|              +--ro ip-prefix-mask                inet:ip-address
|              +--ro output-interface?             string
|              +--ro target-VRF-name?              string
|              +--ro target-nexthop?               inet:ip-address
|              +--ro description?                  string
|              +--ro preference?                   uint32
|              +--ro cost?                         uint32
|              +--ro tag?                          uint32
|              +--ro enable-inherit-cost?          boolean
|              +--ro is-permanent?                 boolean
|              +--ro not-advertise?                boolean
|              +--ro is-iterated-to-host-route?    boolean
|              +--ro classified-pkts?              uint64
|              +--ro classified-bytes?             uint64
+--rw BFD-paras-cfg
   +--rw BFD-enable?          boolean
   +--rw BFD-session-paras* [BFD-session-name]
      +--rw BFD-session-name        string
      +--rw address-family?         identityref
      +--rw interface-name?         string
      +--rw destination-VRF-name?   string
      +--rw nexthop?                inet:ip-address
      +--rw local-ip-address?       inet:ip-address
      +--rw min-tx-interval?        uint32
      +--rw min-rx-interval?        uint32
      +--rw multiplier?             uint32
```

This data model defines the configuration and state containers for static routes. In staticrt-cfg container, there is a list of configuration containers per static route, which contains the configuration for static route.

The data model for state of static routes defines two state containers. Container staticrt-rib contains the current state of static routes. In the second container, there is statistics information for static routes. The staticrt-entries in the staticrt-statistics container is listed per routing instance per address family. The list of staticrt-entries is in order of applied rules in the forwarding path.

This data model also defines the configuration for BFD session using to detect the configured nexthop reachability of the static route.

3. Static Route YANG Module

<CODE BEGINS> file "ietf-staticrt@2015-10-16.yang"

```
module ietf-staticrt{
  namespace "urn:ietf:params:xml:ns:yang:ietf-staticrt";
  prefix staticrt;

  import ietf-inet-types {
    prefix inet;
  }

  organization "TBD";
  contact "TBD";

  description
    "This module contains a collection of YANG definitions for
     configuring static routes.";

  revision 2015-10-16 {
    description
      "Initial revision.";
    reference
      " [draft-ietf-netmod-routing-cfg-16]
       A YANG Data Model for Routing Management.
      ";
  }

  identity address-family {
    description
      "Base identity from which identities describing address
       families are derived.";
  }

  identity ipv4 {
    base address-family;
    description
      "This identity represents IPv4 address family.";
  }

  identity ipv6 {
    base address-family;
    description
      "This identity represents IPv6 address family.";
  }
```

```
     grouping staticrt-entry {
       description
         "This group defines the static route.";

       leaf ip-prefix {
         type inet:ip-address;
         mandatory "true";
         description
           "The destination ip prefix of the static route.";
       }

       leaf ip-prefix-mask {
         type inet:ip-address;
         mandatory "true";
         description
           "The destination ip prefix mask of the
            static route.";
       }

       leaf output-interface {
         type string;
         description
           " The name of the output interface.";
       }

       leaf target-VRF-name {
         type string;
         description
           "The VRF name of the target nexthop.";
       }

       leaf target-nexthop {
         type inet:ip-address;
         description
           " The configured target nexthop.";
       }

       leaf description {
         type string;
         description
           " The description of the static route.";
       }

       leaf preference {
         type uint32;
         description "Specifies route preference.";
       }
```

```
        leaf cost {
          type uint32;
          description "Specifies route cost.";
        }

        leaf tag {
          type uint32;
          description "Specifies route tag.";
        }

        leaf enable-inherit-cost {
          type boolean;
          default false;
          description
            "The flag indicates whether the static route should
             inherit the cost of the iterated route.";
        }

        leaf is-permanent {
          type boolean;
          default false;
          description
            "The flag indicates whether the static route should
             be released permanently.";
        }

        leaf not-advertise {
          type boolean;
          default false;
          description
            "The flag indicates whether the router should not
             advertise the static route.";
        }

        leaf is-iterated-to-host-route {
          type boolean;
          default false;
          description
            "The flag indicates whether the static route should be
             iterated to a host route.";
        }
      }

      grouping staticrt-BFD-paras {
        description
          "This group defines the corresponding BFD session
           information for detecting the configured nexthop
           reachability of the static route.";
```

```
        leaf BFD-enable {
          type boolean;
          description
            " The flag indicates whether the router enable
             the corresponding BFD session for this static route.";
        }

        leaf BFD-session-name {
          type string;
          description "The BFD session name.";
        }

        leaf adminstated-down-not-selected {
          type boolean;
          description
            "If this flag is true,it indicates that the corresponding
             static route should not be selected when the BFD session
             associated with it is in the AdminDown State.";
        }
      }

      grouping staticrt-reachability-paras {
        description
          "This group defines the reachability detecting session
           information for detecting the configured nexthop
           reachability of the static route.";

        uses staticrt-BFD-paras;
      }

      grouping staticrt-BFD-state {
        description
          "This group defines the corresponding BFD session state
           as a detecting result of the configured nexthop
           reachability.";

        leaf BFD-state {
          type string;
          config false;
          description "The BFD session state.";
        }

        leaf local-address {
          type inet:ip-address;
          config false;
          description
            "the local IP address of the corresponding BFD session.";
        }
```

```
        leaf remote-address {
          type inet:ip-address;
          config false;
          description
            "the remote IP address of the corresponding BFD session.";
        }
      }

      grouping staticrt-reachability-state {
        description
          "This group defines the reachability detecting session
           state as a detecting result of the configured nexthop
           reachability.";

        uses staticrt-BFD-state;
      }

      container staticrt {
        description
          "Container for static route configuration and state";
        container staticrt-cfg {
          description
            "Configuration for static route.";
          list staticrt-topo {
            key "VRF-name address-family topo-name";
            description
              "Configuration of a static route list.";

            leaf topo-name {
              type string;
              description
                "The topology name of the destination ip prefix
                 belonging to.";
            }
            leaf VRF-name {
              type string;
              description
                "The VRF-name of the staticrt-entry";
            }
            leaf address-family {
              type identityref {
                base address-family;
              }
              description
                "address-family of the staticrt-entry";
            }

            leaf enable-FRR {
```

```
            type boolean;
            description
              "Enable FRR.FRR is implemented only on static routes
               that are manually configured. That is, FRR is not
               implemented on iterated next hops.";
          }

          list staticrt-entries {
            key "ip-prefix ip-prefix-mask output-interface "
               +"target-VRF-name target-nexthop";
            ordered-by system;

            uses staticrt-entry;
            uses staticrt-reachability-paras;

            description
              "Define static routes.";
          }
        }
      }
      container staticrt-state {
        config false;
        description
           "Operational state of static routes.";

        container staticrt-rib{
          description
            "Define the operational state data for static routes.";

          list staticrt-route {
            description
              "Static routes are organized into list of routes.";

            leaf index {
              type uint32;
              description
                "Static route entry index.";
            }

            leaf topo-name {
              type string;
              description
                "The topology name of the destination ip prefix
                 belonging to.";
            }

            leaf VRF-name {
              type string;
```

```
            description
              "VRF-name of the set of static routes";
          }

          leaf address-family {
            type string;
            description
              "Address-family of the set of static-routes";
          }

          uses staticrt-entry;
          uses staticrt-reachability-paras;

          leaf iterated-nexthop {
            type inet:ip-address;
            config false;
            description "The iterated nexthop.";
          }

          leaf iterated-output-interface {
            type string;
            config false;
            description "The iterated output interface name.";
          }

          leaf label {
            type uint32;
            config false;
            description "Specifies egress label.";
          }

          leaf route-state {
            type string;
            config false;
            description "Route state.";
          }

          leaf interface-state {
            type string;
            config false;
            description "The configured output interface state.";
          }
          uses staticrt-reachability-state;
        }
      }
      container staticrt-statistics {
        config false;
        description
```

```
            "Define the statistics of list of static routes.";
          list staticrt-stats {
            description
              "Statistics of list of static routes per VRF &
               Address-family & topology.";

            leaf topo-name {
              type string;
              description
                "The topology name of the destination ip prefix
                 belonging to.";
            }

            leaf VRF-name {
              type string;
              description
                "Vrf-name of the set of static routes";
            }

            leaf address-family {
              type string;
              description
                "Address-family of the set of static-routes";
            }

            list static-route-stats {
              description
                "This defines the static route statistics of
                 each static route.";

              uses staticrt-entry;

              leaf classified-pkts {
                type uint64;
                description
                  " Number of total packets which matched
                    to the static route";
              }
              leaf classified-bytes {
                type uint64;
                description
                  " Number of total bytes which matched
                    to the static route";
              }
            }
          }
        }
      }
```

```
     }

     container BFD-paras-cfg {
       description
         "This container defines the corresponding BFD session
          information for detecting the configured nexthop
          reachability of the static route.";

       leaf BFD-enable {
         type boolean;
         description
           "The flag indicates whether the router enable
            the corresponding BFD session for this static route.";
       }

       list BFD-session-paras {
         key "BFD-session-name";
         description "The BFD sessions for static routes";

         leaf BFD-session-name {
           type string;
           description "The BFD session name.";
         }

         leaf address-family {
           type identityref {
             base address-family;
           }
           description
             " address-family of the staticrt-entry";
         }

         leaf interface-name {
           type string;
           description "Interface name.";
         }

         leaf destination-VRF-name {
           type string;
           description "Destination vpn instance name for Gateway.";
         }

         leaf nexthop {
           type inet:ip-address;
           description "NextHop address.";
         }

         leaf local-ip-address {
```

```
          type inet:ip-address;
          description "The local ip address of the BFD session.";
        }

        leaf min-tx-interval {
          type uint32;
          description "Min transmit interval.";
        }

        leaf min-rx-interval {
          type uint32;
          description "Min receive interval.";
        }

        leaf multiplier {
          type uint32;
          description "Multiplier value.";
        }
      }
    }
  }
```

<CODE ENDS>

4. IANA Considerations

This document registers a URI in the "IETF XML Registry" [RFC3688]. Following the format in [RFC3688], the following registration has been made.

URI: urn:ietf:params:xml:ns:yang:ietf-staticrt

Registrant Contact: The RTGWG WG of the IETF.

XML: N/A; the requested URI is an XML namespace.

This document registers a YANG module in the "YANG Module Names" registry [RFC6020].

Name: ietf-staticrt

Namespace: urn:ietf:params:xml:ns:yang:ietf-staticrt

Prefix: staticrt

Reference: RFC xxxx

5. Security Considerations

The YANG module defined in this memo is designed to be accessed via the NETCONF protocol [RFC6241]. The lowest NETCONF layer is the secure transport layer and the mandatory-to-implement secure transport is SSH [RFC6242]. The NETCONF access control model [RFC6536] provides the means to restrict access for particular NETCONF users to a pre-configured subset of all available NETCONF protocol operations and content.

There are a number of data nodes defined in the YANG module which are writable/creatable/deletable (i.e., config true, which is the default). These data nodes may be considered sensitive or vulnerable in some network environments. Write operations (e.g., <edit-config>) to these data nodes without proper protection can have a negative effect on network operations.

6. Acknowledgements

The editor of this document wishes to thank Eric Wu, Guangying Zheng, Baohua Song, Xiangfeng Ding for the guidance and support in coming up with this draft.

7. References

7.1. NormativeInformative

[RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <http://www.rfc-editor.org/info/rfc3688>.

[RFC6020] Bjorklund, M., Ed., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, DOI 10.17487/RFC6020, October 2010, <http://www.rfc-editor.org/info/rfc6020>.

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

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

[RFC6536] Bierman, A. and M. Bjorklund, "Network Configuration Protocol (NETCONF) Access Control Model", RFC 6536, DOI 10.17487/RFC6536, March 2012, <http://www.rfc-editor.org/info/rfc6536>.

7.2. Informative References

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

Authors' Addresses

Qiandeng Liang
Huawei Technologies
101 Software Avenue, Yuhuatai District
Nanjing 210012
China

Email: liangqiandeng@huawei.com

Gang Yan
Huawei Technologies
Huawei Bld., No.156 Beiqing Rd.
Beijing 100095
China

Email: yangang@huawei.com

Shunwan Zhuang
Huawei Technologies
Huawei Bld., No.156 Beiqing Rd.
Beijing 100095
China

Email: zhuangshunwan@huawei.com

Network Working Group
Internet Draft
Intended status: Proposed Standard
Expires: April 13, 2016

Y. Liu
Q. Chen
A. Foldes
Ericsson
October 13, 2015

Yang Data Model for GRE Tunnel
draft-liu-intarea-gre-tunnel-yang-00.txt

Status of this Memo

Copyright Notice

Abstract

This document defines a YANG data model for the management of GRE tunnels. The data model covers configuration data and operational state data.

Table of Contents

1. Introduction

This document defines a YANG [RFC6020] data model for the management of GRE tunnels. It covers the following types.

1.1. Terminology

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].

1.2. Tree Diagrams

A simplified graphical representation of the data model is used in this document. The meaning of the symbols in these diagrams is as follows:

- Brackets "[" and "]" enclose list keys.
- Abbreviations before data node names: "rw" means configuration (read-write), and "ro" means state data (read-only).
- Symbols after data node names: "?" means an optional node, "!" means a presence container, and "*" denotes a list and leaf-list.
- Parentheses enclose choice and case nodes, and case nodes are also marked with a colon (":").

o Ellipsis ("...") stands for contents of subtrees that are not shown.

2. GRE Tunnel Data Model

This document defines the YANG model "ietf-gre-tunnel", which includes two modules, one for configuration and one for state. The data model has the following tree diagram for the GRE tunnels:

```
module: ietf-gre-tunnel

   +--rw gre-tunnels

   |  +--rw gre* [name]

   |     +--rw name                   string

   |     +--rw description?           string

   |     +--rw bind-interface?        if:interface-ref

   |     +--rw clear-df?              empty

   |     +--rw keepalive

   |     |  +--rw interval?    uint32

   |     |  +--rw retry-num?   uint32

   |     +--rw mirror-destination?   string

   |     +--rw mtu?                   uint16

   |     +--rw shutdown?              empty

   |     +--rw hop-limit?             uint8

   |     +--rw tos?                   int8

   |     +--rw peer-end-point

   |     |  +--rw local?               inet:ipv4-address-no-zone

   |     |  +--rw remote?              inet:ipv4-address-no-zone

   |     |  +--rw routing-instance?   rt:routing-instance-ref
```

```
|     +--rw tunnel-circuits* [key-id]
|        +--rw key-id                uint32
|        +--rw description?          string
|        +--rw bind-interface?       if:interface-ref
|        +--rw clear-df?             empty
|        +--rw keepalive
|        |  +--rw interval?    uint32
|        |  +--rw retry-num?   uint32
|        +--rw mirror-destination?   string
|        +--rw mtu?                  uint16
|        +--rw shutdown?             empty
|        +--rw hop-limit?            uint8
|        +--rw tos?                  int8
+--ro tunnel-state
   +--ro gre*
      +--ro name?                      string
      +--ro local-ip?                  inet:ipv4-address-no-zone
      +--ro remote-ip?                 inet:ipv4-address-no-zone
      +--ro state?                     enumeration
      +--ro bind-interface?            if:interface-state-ref
      +--ro routing-instance?          rt:routing-instance-ref
      +--ro mtu?                       uint16
      +--ro clear-df?                  empty
      +--ro tunnel-id?                 uint32
```

```
            +--ro down-reason?               string
            +--ro resolved-interface-name?   string
            +--ro hop-limit?                 uint8
            +--ro tos?                       int8
            +--ro keepalive
            |  +--ro interval?    uint32
            |  +--ro retry-num?   uint32
            +--ro tunnel-circuits*
               +--ro key-id?                    uint32
               +--ro name?                      string
               +--ro local-ip?                  inet:ipv4-address-no-
   zone
               +--ro remote-ip?                 inet:ipv4-address-no-
   zone
               +--ro state?                     enumeration
               +--ro bind-interface?            if:interface-state-ref
               +--ro routing-instance?          rt:routing-instance-ref
               +--ro mtu?                       uint16
               +--ro clear-df?                  empty
               +--ro tunnel-id?                 uint32
               +--ro down-reason?               string
               +--ro resolved-interface-name?   string
               +--ro hop-limit?                 uint8
               +--ro tos?                       int8
               +--ro keepalive
```

```
                +--ro interval?    uint32
                +--ro retry-num?   uint32
   augment /if:interfaces-state/if:interface:
      +--ro gre-tunnel-enabled?   boolean
```

3. GRE Tunnel YANG Model

```
   <CODE BEGINS>
   module ietf-gre-tunnel {

     namespace "urn:ietf:params:xml:ns:yang:ietf-gre-tunnel";
     prefix "gretln";

     import ietf-interfaces {
       prefix "if";
     }

     import ietf-inet-types {
       prefix inet;
     }

     import ietf-routing {
       prefix "rt";
     }

     organization
```

```
       "Ericsson.";

     contact
       "Mandy.Liu@ericsson.com
      Adam.Foldes@ericsson.com";

     description
       "This YANG model defines the configuration data
      and operational state data for GRE tunnel.";

     revision 2015-10-13 {
       description
         "Initial revision.";
       reference
         "RFC XXXX: A YANG Data Model for GRE Tunnel.";
     }


     grouping tunnel-gre-components {
       description
         "This grouping provides common attributes for
          GRE tunnels and tunnel circuits.";
       leaf description {
         type string;
```

```
          description
            "Textual description for a tunnel. Can be any
          alphanumeric string, including spaces, not to
          exceed 255 ASCII characters.";
        }
        leaf bind-interface {
          type if:interface-ref;
          description
             "Bind to an interface.";
        }
        leaf clear-df {
          type empty;
          description
            "If clear-df is absent, it means that fragmentation
          of tunnel packets are permitted. If clear-df is
          present, it means that fragmentation of tunnel packets
          are not permitted.";
        }
        container keepalive {
          description
            "Enables sending keepalive packets on GRE tunnels,
             and specifies the interval and number of retries.";
          leaf interval {
            type uint32 {
```

```
            range "5..40";
          }
          units "seconds";
          default "10";
          description
            "Number of seconds between sending keepalive packets.";
        }
        leaf retry-num {
          type uint32 {
            range "2..255";
          }
          default "4";
          description
            "Number of times a keepalive packet is sent without
             response before the tunnel is brought down.";
        }
      }
      leaf mirror-destination {
        type string;
        description
          "Designate the name of a tunnel as a circuit
        mirror destination. ";
      }
      leaf mtu {
```

```
          type uint16 {
            range "256..16384";
          }
          description
            "Sets the Maximum Transmission Unit (MTU) size for
            packets sent in a tunnel. The default MTU is the MTU
            for the interface to which the tunnel is bound.";
        }
        leaf shutdown {
          type empty;
          description
            "Disable/enable the tunnel.";
        }
        leaf hop-limit {
         type uint8 {
           range "0|1..255";
         }
         description
           "The IPv4 TTL or IPv6 Hop Limit which is used in the outer IP
           header. A value of 0 indicates that the value is copied from
           the payload's header.";
       }
       leaf tos {
         type int8 {
```

```
          range "-1..63";
        }
        description
          "The method used to set the high 6 bits (the differentiated
         services codepoint) of the IPv4 TOS or IPv6 Traffic Class in
         the outer IP header. A value of -1 indicates that the bits are
         copied from the payload's header. A value between 0 and 63
         inclusive indicates that the bit field is set to the indicated
         value.";
      }
     }

     /*Configuration Data*/
     container gre-tunnels {
       description
        "Configuration data for tunnels.";
       list gre {
         key "name";
         description
           "Configuration of GRE tunnel.";
         leaf name {
           type string;
           description
```

```
            "Name of the tunnel.";
          }
          uses tunnel-gre-components;
          container peer-end-point {
            description
              "Assigns IP addresses to tunnel endpoints.";
            leaf local {
              type inet:ipv4-address-no-zone;
              description
                "IP address of the local end of the tunnel.";
            }
            leaf remote {
              type inet:ipv4-address-no-zone;
              description
                "IP address of the remote end of the tunnel.";
            }
            leaf routing-instance {
              type rt:routing-instance-ref;
              description
                "Name of the reference routing instance.";
            }
          }
          list tunnel-circuits {
            key "key-id";
```

```
            description
              "Configuration of GRE tunnel circuit.";
            leaf key-id {
              type uint32;
              description
                "Specifies a key ID in the current GRE tunnel.";
            }
            uses tunnel-gre-components;
          }
        }
      }

      /*Operational state data*/
      grouping tunnel-gre-states {
        description
         "The basic tunnel information to be displayed.";
        leaf name {
          type string;
          description
            "Name of the tunnel.";
        }
        leaf local-ip {
          type inet:ipv4-address-no-zone;
          description
```

```
            "IP address of the local end of the tunnel.";
        }
        leaf remote-ip {
          type inet:ipv4-address-no-zone;
          description
            "IP address of the remote end of the tunnel.";
        }
        leaf state {
          type enumeration {
            enum Down {
            description
              "Tunnel down state.";
            }
            enum Up {
              description
                "Tunnel up state.";
            }
            enum Shutdown {
              description
                "Tunnel shutdown state.";
            }
            enum Keep-down {
              description
                "Tunnel keepalive down state.";
```

```
            }
            enum Wait-on-SA {
              description
                "Tunnel pending SA UP state.";
            }
            enum Not-used {
              description
                "Tunnel is not in used.";
            }
          }
          description
            "Indicates the state of the tunnel.";
        }
        leaf bind-interface {
          type if:interface-state-ref;
          description
            "The name of the interface to which the tunnel is bound.";
        }
        leaf routing-instance {
          type rt:routing-instance-ref;
          description
            "Indicates the name of the reference routing instance.";
        }
        leaf mtu {
```

```
          type uint16;
          description
            "The Maximum Transmission Unit (MTU) size for
          packets sent in a tunnel.";
        }
        leaf clear-df {
          type empty;
          description
            "Indicate that the DF bit is cleared.";
        }
        leaf tunnel-id {
          type uint32;
          description
            "Tunnel id.";
        }
        leaf down-reason {
          type string;
          description
            "Indicate the down reason of the tunnel.";
        }
        leaf resolved-interface-name{
          type string;
          description
            "The egress interface name of the tunnel.";
```

```
       }
      leaf hop-limit {
        type uint8;
        description
          "The IPv4 TTL or IPv6 Hop Limit which is used in the outer IP
         header. A value of 0 indicates that the calue is copied from
         the payload's header.";
      }
      leaf tos {
        type int8;
        description
          "The high 6 bits (the differentiated
         services codepoint) of the IPv4 TOS or IPv6 Traffic Class in
         the outer IP header. A value of -1 indicates that the bits are
         copied from the payload's header. A value between 0 and 63
         inclusive indicates that the bit field is set to the indicated
         value.";
      }
     }

     container tunnel-state {
       config "false";
       description
         "Contain the information currently configured tunnels.";
```

```
      list gre {
        description
          "Operational state data of GRE tunnel.";
        uses tunnel-gre-states;
        container keepalive {
          description
            "The interval and number of retries for
             sending keepalive packets on GRE tunnels.";
          leaf interval {
            type uint32;
            units "seconds";
            description
              "Number of seconds between sending keepalive packets.";
          }
          leaf retry-num {
            type uint32;
            description
              "Number of times a keepalive packet is sent without
               response before the tunnel is brought down.";
          }
        }
        list tunnel-circuits {
          description
```

```
            "Operational state data of GRE tunnel circuit.";
          leaf key-id {
            type uint32;
            description
              "Key ID of the GRE tunnel circuit.";
          }
          uses tunnel-gre-states;
          container keepalive {
            description
              "The interval and number of retries for
               sending keepalive packets on GRE tunnels.";
            leaf interval {
              type uint32;
              units "seconds";
              description
                "Number of seconds between sending keepalive
   packets.";
            }
            leaf retry-num {
              type uint32;
              description
                "Number of times a keepalive packet is sent without
                 response before the tunnel is brought down.";
            }
```

```
          }
        }
      }
    }

    //Augment operational state data of IP interfaces
    augment "/if:interfaces-state/if:interface" {
      when "if:type = 'ianaift:tunnel'" {
        description
          "Augment IP interface.";
      }
      description
        "Augment operational state data of IP interfaces.";
      leaf gre-tunnel-enabled {
        type boolean;
       description
          "Indicate the type of the IP tunnel interface.
          TRUE means GRE tunnel interface.";
      }
    }
  }// end of module ietf-gre-tunnel
  <CODE ENDS>
```

4. Security Considerations

This document does not introduce any new security risk.

5. IANA Considerations

This document makes no request of IANA.

6. Acknowledgements

The authors would like to thank Xufeng Liu, In-Wher Chen for their contributions to this work.

7. References

7.1. Normative References

[RFC1981] J. McCann, "Path MTU Discovery for IP version 6", RFC 1981, August 1996.

[RFC1702] S. Hanks, "Generic Routing Encapsulation over IPv4 networks", RFC 1702, October 1994.

[RFC2784] D. Farinacci, "Generic Routing Encapsulation", RFC 2784, March 2000.

[RFC2893] R. Gilligan, "Transition Mechanisms for IPv6 Hosts and Routers", RFC 2893, August 2000.

[RFC6020] Bjorklund, M., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, October 2010.

7.2. Informative References

[RFC6241] Enns, R., Bjorklund, M., Schoenwaelder, J., and A. Bierman, "Network Configuration Protocol (NETCONF)", RFC 6241, June 2011.

[RFC7223] Bjorklund, M., "A YANG Data Model for Interface Management", RFC 7223, May 2014.

Authors' Addresses

Ying Liu
Ericsson
No.5 Lize East Street
Beijing, 100102
China

Email: Mandy.Liu@ericsson.com

Qiang Chen
Ericsson
No.5 Lize East Street
Beijing, 100102
China

Email: Qiang.Chen@ericsson.com

Adam Mate Foldes
Ericsson
300 Holger Way
San Jose, CA 95134
USA

Email: Adam.Foldes@ericsson.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: January 4, 2017

Y. Liu
A. Foldes
Ericsson
G. Zheng
Z. Wang
Y. Zhuang
Huawei
A. Wang
China Telecom
July 3, 2016

Yang Data Model for IPIPv4 Tunnel
draft-liu-intarea-ipipv4-tunnel-yang-02

Abstract

This document defines a YANG data model for the management of IP based tunnels. The data model includes configuration data and state data. In default format, it describes managed attributes used for managing tunnels of IPv4 or IPv6 over IPv4 tunnels. And it can also serve as a base model which can be augmented with technology-specific details in other Ip based tunnel models.

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 January 4, 2017.

Table of Contents

1. Introduction

An overview of tunnel is presented at [intarea-tunnel]. Over the past several years, there has been a number of "tunneling" protocols specified by the IETF. And this document defines a YANG data model for the management of IP bases tunnels. In default format, it covers the following tunnel types:

o IPv4 in IPv4, related concepts are defined in [RFC1853]

o IPv6 in IPv4 manual tunnel, related concepts are defined in [RFC2003]

o IPv6 to IPv4 tunnel, related concepts are defined in [RFC3056]

And notice that this model provides a framework and some reusable common attributes where technology-specific IP tunnel YANG models can inherit constructs from it without needing to redefine them within the sub-technology. Therefore it also can serve as a base model which can be extended to include technology specific details.

1.1. Terminology

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].

1.2. Tree Diagrams

A simplified graphical representation of the data model is used in this document. The meaning of the symbols in these diagrams is as follows:

o Brackets "[" and "]" enclose list keys.

o Abbreviations before data node names: "rw" means configuration (read-write), and "ro" means state data (read-only).

o Symbols after data node names: "?" means an optional node, "!" means a presence container, and "*" denotes a list and leaf-list.

o Parentheses enclose choice and case nodes, and case nodes are also marked with a colon (":").

o Ellipsis ("...") stands for contents of subtrees that are not shown.

2. Architecture of IP tunnel YANG Model

In this document we define an IP based tunnel model, in default it can describe the IPv6/4-in-IPv4 tunnels including IPv4 in IPv4 [RFC1853], IPv6 in IPv4 [RFC2003], IPv6 to IPv4 [RFC3056].

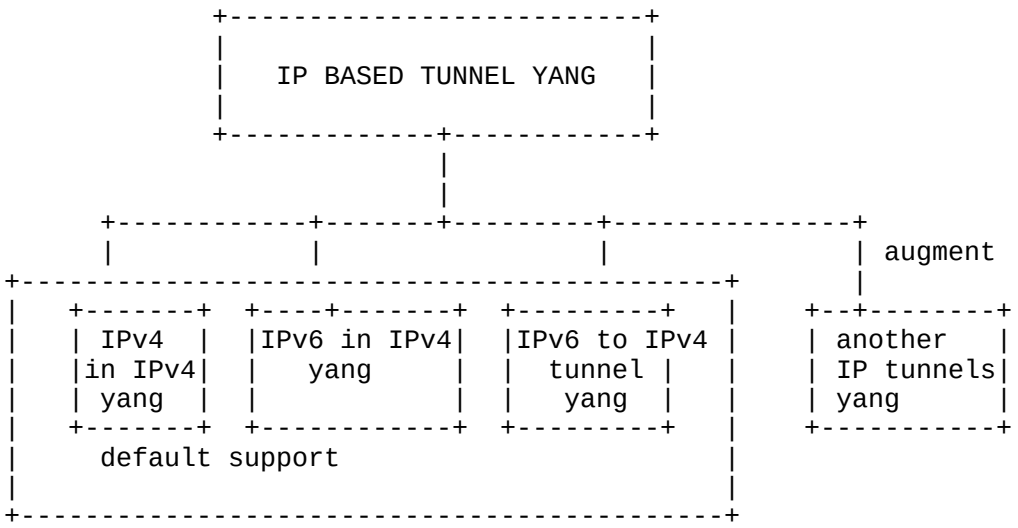
Since some reusable common core attributes of ip tunnels are defined, it can also provide an optional solution for extending to the other IP-based tunnel models. The users of IP-based tunnel model can according to following method to define other technologies specific ip tunnel models:

o The IP based Tunnel YANG model can acts as the root for other Tunnel YANG models.

o Augment the ip based tunnel model by adding the tunnel type is defined as an identity that augments the base " ip-tunnel-type " defined in the IP based model.

o Augment the ip based tunnel model by adding new data nodes with technology specific parameters into proper anchor points of the ip tunnel model.

Figure 1 depicts the relationship of different Tunnel YANG models to the Ip Tunnel YANG Model. It can default support the IPv4-in-IPv4, IPv6-in-IPv4, and IPv6-to-IPv4, and can also be extended to another IP base tunnels encapsulation protocol.



Relationship of technology specific TUNNEL YANG model to IP based tunnel YANG model

3. IP Tunnel Model Design

3.1. Model Overview

This document defines the YANG model "ietf-ip-tunnel". It includes two modules, one for configuration and one for state. At the top of the Model there is Tunnels container for tunnel configuration data. Multiple Tunnel lists keyed using tunnel name and tunnel type within the Tunnels container. To correctly identify an ip based tunnel the bind-interface, local-address, remote-address are defined under one tunnel list. Notice that tunnels are handled by creating a logical interface for each tunnel. Each logical interface (physical or virtual) need to map to interface yang model [RFC7223]. To do so, in this draft, the bind-interface are defined which with a leafref type and it can be used pointing to the corresponding interface-name node in the interface yang [RFC7223].

The data model also includes read-only counter so that the tunnel state can be read.

3.2. IP Tunnel YANG Tree Diagrams

The data model has the following tree diagram for the IP tunnels:

```
module: ietf-ip-tunnel
   +--rw Tunnels
   |  +--rw Tunnel* [name type]
   |     +--rw name                   string
   |     +--rw type                   identityref
   |     +--rw local-address?         inet:ip-address-no-zone
   |     +--rw remote-address?        inet:ip-address-no-zone
   |     +--rw routing-instance?      routing-instance-ref
   |     +--rw description?           string
   |     +--rw bind-interface?        if:interface-ref
   |     +--rw clear-df?              empty
   |     +--rw shutdown?              empty
   |     +--rw tmtu?                  uint16
   |     +--rw mirror-destination?    string
   |     +--rw hop-limit?             uint8
   |     +--rw tos?                   int8
   +--ro tunnel-state
      +--ro tunnels*
         +--ro name?                        string
         +--ro type?                        identityref
         +--ro local-ip?                    inet:ip-address-no-zone
         +--ro remote-ip?                   inet:ip-address-no-zone
         +--ro (state)?
         |  +--:(up)
         |  |  +--ro up?                          empty
         |  +--:(down)
         |  |  +--ro down?                        empty
         |  +--:(shutdown)
         |     +--ro shutdown?                    empty
         +--ro bind-interface?              if:interface-state-ref
         +--ro user-configured?             boolean
         +--ro routing-instance?            routing-instance-ref
         +--ro tmtu?                        uint16
         +--ro clear-df?                    empty
         +--ro down-reason?                 string
         +--ro resolved-interface-name?     string
         +--ro hop-limit?                   uint32
         +--ro tos?                         int32
augment /if:interfaces-state/if:interface:
   +--ro tunnel-protocol?   identityref
```

4. IP Tunnel YANG Data Model

```
<CODE BEGINS> file "ietf-ip-tunnel@2016-06-20.yang"
module ietf-ip-tunnel{

  namespace "urn:ietf:params:xml:ns:yang:ietf-ip-tunnel";
  prefix "iptnl";

  import ietf-interfaces {
    prefix "if";
  }

  import ietf-inet-types {
    prefix inet;
  }

  import ietf-routing {
    prefix "rt";
  }


  import iana-if-type {
    prefix ianaift;
  }

  organization
    "IETF NETMOD (NETCONF Data Modeling Language) Working Group.";

  contact
    "Mandy.Liu@ericsson.com
     Adam.Foldes@ericsson.com
     zhengguangying@huawei.com";

  description
    "This YANG model defines the configuration data
     and operational state data for generic IPv4/6-in-IPv4 tunnel.
     It includes the IPv4 in IPv4, 6-to-4, and IPv6 over IPv4 manual
     tunnels.";

  revision 2016-04-27 {
    description
      "Made model more generic in order to allow augmentation by e.g.
       GRE tunnels.";
    reference
      "RFC XXXX: A YANG Data Model for IPv4 Tunnel.";
  }
  revision 2016-03-11 {
    description
```

```
      "Collapsed all tunnel types into a single subtree based on
       suggestions to more closely follow the IP Tunnel MIB.";
    reference
      "RFC XXXX: A YANG Data Model for IPv4 Tunnel.";
  }
  revision 2015-10-14 {
    description
      "Update model based on comments.";
    reference
      "RFC XXXX: A YANG Data Model for IPv4 Tunnel.";
  }

  revision 2015-07-20 {
    description
      "This version adds the following new items:
        - hop-limit
        - tos
        - tunnel-type
       This version changes 'ipv6to4-auto' to 'ipv6to4'";
    reference
      "RFC XXXX: A YANG Data Model for IPv4 Tunnel.";
  }

  revision 2015-05-27 {
    description
      "Initial revision.";
    reference
      "RFC XXXX: A YANG Data Model for IPv4 Tunnel.";
  }

  /* Identities */
  identity ip-tunnel-type {
    description
      "Base identity from which identities describing
       IP tunnel types are derived.";
  }
  identity ip-ip {
    base ip-tunnel-type;
    description
      "This identity represents IPv4-in-IPv4 tunnel type";
  }
  identity ipv6v4-manual {
    base ip-tunnel-type;
    description
      "This identity represents IPv6-to-IPv4 manual tunnel type";
  }
  identity ipv6-to-v4 {
    base ip-tunnel-type;
```

```
    description
      "This identity represents the 6-to-4 tunnel type";
  }

        typedef routing-instance-ref {
        type leafref {
          path "/rt:routing/rt:routing-protocols/rt:routing-protocol/rt:name";
        }
        description
        "This type is used for leafs that reference a routing instance
         configuration.";
      }

  /*Configuration Data*/
  container Tunnels{
    description
     "Configuration data for tunnels.";
    list Tunnel{
      key "name type";
      description
        "Configuration data for tunnels.";
      leaf name {
        type string;
        description
          "Name of the tunnel.";
      }
      leaf type {
        type identityref {
          base ip-tunnel-type;
        }
        description "The encapsulation type of the tunnel.";
      }

      leaf routing-instance {
        type routing-instance-ref;
        description "The routing instance of the local address.";
      }
      uses tunnel-config-components;
    }
  }

  /* Configuration data */
  grouping tunnel-config-components {
      description
        "Configuration data for all tunnels and subtunnels";
      leaf description {
        type string {
          length "1..255";
```

```
        }
        description
          "Textual description for a tunnel. Can be any "+
          "alphanumeric string, including spaces, not to exceed "+
          "255 ASCII characters.";
      }
      leaf bind-interface {
        type if:interface-ref;
        description
          "Bind to an interface.";
      }
          leaf local-address {
        type inet:ip-address-no-zone;
        description "IP address of the local end of the tunnel.";
      }
      leaf remote-address {
        when "type != ipv6-to-v4" {
          description
            "6-to-4 tunnels do not have a fixed remote endpoint.";
        }
        type inet:ip-address-no-zone;
        description "IP address of the remote end of the tunnel.";
      }

      leaf clear-df {
        type empty;
        description
          "If clear-df is absent, it means that fragmentation of
           tunnel packets are permitted. If clear-df is present,
           it means that fragmentation of tunnel packets are not
           permitted.";
      }
      leaf shutdown {
        type empty;
        description
          "Disable/enable the tunnel.";
      }
      leaf tmtu {
        type uint16 {
          range "256..16384";
        }
        description
          "Sets the Maximum Transmission Unit (MTU) size for
           packets sent in a tunnel. The default MTU is the MTU
           for the interface to which the tunnel is bound.";
      }
      leaf mirror-destination {
        type string;
```

```
        description
          "Designate the name of a tunnel as a circuit
           mirror destination. ";
      }
      leaf hop-limit {
        type uint8 {
          range "0|1..255";
        }
        description
          "The IPv4 TTL or IPv6 Hop Limit which is used in the
           outer IP header. A value of 0 indicates that the value
           is copied from the payload's header.";
      }
      leaf tos {
        type int8 {
          range "-1..63";
        }
        description
          "The method used to set the high 6 bits (the
            differentiated services codepoint) of the IPv4 TOS or
            IPv6 Traffic Class in the outer IP header. A value of -1
            indicates that the bits are copied from the payload\u2019s
            header. A value between 0 and 63 inclusive indicates
            that the bit field is set to the indicated value.";
      }

  }

  /*Operational state data*/
  grouping tunnel-oper-states {
    description "Operational states of tunnels";
    choice state {
      description "Choice of operational states";
      case up {
        leaf up {
          type empty;
          description "The tunnel is up.";
        }
      }
      case down {
        leaf down {
          type empty;
          description "The tunnel is down.";
        }
      }
      case shutdown {
        leaf shutdown {
          type empty;
```

```
          description "The tunnel is shut down administratively.";
        }
      }
    }
  }

  grouping tunnel-state-components {
    description
      "The basic tunnel information to be displayed.";

    leaf name {
      type string;
      description
        "Name of the tunnel.";
    }

    leaf type {
      type identityref;
      description
        "The type of the tunnel.";
    }
    leaf local-ip {
      type inet:ip-address-no-zone;
      description
        "IP address of the local end of the tunnel.";
    }
    leaf remote-ip {
      type inet:ip-address-no-zone;
      description
        "IP address of the remote end of the tunnel.";
    }
    uses tunnel-oper-states;
    leaf bind-interface {
      type if:interface-state-ref;
      description
        "Bind to an interface.";
    }
    leaf user-configured {
      type boolean;
          description
        "Indicate the tunnel is user-configured or dynamic.
        False is for dynamic.";
    }
    leaf routing-instance {
      type routing-instance-ref;
      description
        "Name of the reference routing instance. ";
    }
```

```
    leaf tmtu {
      type uint16;
      description
        "The Maximum Transmission Unit (MTU) size for
      packets sent in a tunnel.";
    }
    leaf clear-df {
      type empty;
      description
        "Indicate that the DF bit is cleared.";
    }
    leaf down-reason {
      type string;
      description
        "The reason of the tunnel is down.";
    }
    leaf resolved-interface-name{
      type string;
      description
        "The egress interface name of the tunnel.";
    }
    leaf hop-limit {
      type uint32;
      description
        "The IPv4 TTL or IPv6 Hop Limit which is used in the outer IP
         header. A value of 0 indicates that the calue is copied from
         the payload's header.";
    }
    leaf tos {
      type int32;
      description
        "The high 6 bits (the differentiated
         services codepoint) of the IPv4 TOS or IPv6 Traffic Class in
         the outer IP header. A value of -1 indicates that the bits
         are copied from the payload\u2019s header. A value between 0 and
         63 inclusive indicates that the bit field is set to the
         indicated value.";
    }
  }

  container tunnel-state {
    config "false";
      description
        "Contain the information currently configured tunnels.";
      list tunnels {
        description
           "Operational state data of tunnels.";
         uses tunnel-state-components;
```

```
      }
  }

  //Augment operational state data of IP interfaces
  augment "/if:interfaces-state/if:interface" {
    when "if:type = 'ianaift:tunnel'" {
      description
        "Augment IP interface.";
    }
    description
      "Augment operational state data of IP interfaces.";
    leaf tunnel-protocol {
      type identityref;
      description
        "Indicate the state of the IP tunnel interface.";
    }
  }
}
```

<CODE ENDS>

5. Security Considerations

The YANG module defined in this memo is designed to be accessed via the NETCONF protocol [RFC6241] . The lowest NETCONF layer is the secure transport layer and the mandatory-to-implement secure transport is SSH [RFC6242] . The NETCONF access control model [RFC6536] provides the means to restrict access for particular NETCONF users to a pre-configured subset of all available NETCONF protocol operations and content.

6. IANA Considerations

This document registers a URI in the IETF XML registry [RFC3688] . Following the format in RFC 3688, the following registration is requested to be made:

URI: urn:ietf:params:xml:ns:yang:ietf-ip-tunnel

Registrant Contact: The IESG.

XML: N/A, the requested URI is an XML namespace.

This document registers a YANG module in the YANG Module Names registry [RFC6020].

name: ietf-tunnel

namespace:urn:ietf:params:xml:ns:yang:ietf-ip-tunnel

prefix: itun reference: RFC XXXX

7. Normative References

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

[RFC6020] Bjorklund, M., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, October 2010.

Authors' Addresses

Ying Liu
Ericsson
No.5 Lize East Street
Beijing 100102
China

Email: Mandy.Liu@ericsson.com

Adam Mate Foldes
Ericsson
300 Holger Way
San Jose CA 95134
USA

Email: Adam.Foldes@ericsson.com

Guangying Zheng
Huawei Technologies,Co.,Ltd
101 Software Avenue, Yuhua District
Nanjing 210012
China

Email: zhengguangying@huawei.com

Zitao Wang
Huawei Technologies,Co.,Ltd
101 Software Avenue, Yuhua District
Nanjing 210012
China

Email: wangzitao@huawei.com

Yan Zhuang
Huawei
101 Software Avenue, Yuhua District
Nanjing, Jiangsu 210012
China

Email: zhuangyan.zhuang@huawei.com

Aijun Wang
China Telecom
No.118,Xizhimenneidajie,Xicheng District
Beijing 100035
China

Email: wangaj@ctbri.com.cn

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: October 5, 2016

X. Liu, Editor
A. Kyparlis
R. Parikh
Ericsson
A. Lindem
Cisco Systems
M. Zhang
Huawei Technologies

April 5, 2016

A YANG Data Model for Virtual Router Redundancy Protocol (VRRP)
draft-liu-rtgwg-yang-vrrp-04.txt

Abstract

This document describes a data model for Virtual Router Redundancy Protocol (VRRP). Both version 2 and version 3 of VRRP are covered.

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), 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/1id-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 October 5, 2016.

Copyright Notice

Table of Contents

1. Introduction

This document introduces a YANG [RFC6020] data model for Virtual Router Redundancy Protocol (VRRP) [RFC3768][RFC5798]. VRRP provides higher resiliency by specifying an election protocol that dynamically assigns responsibility for a virtual router to one of the VRRP routers on a LAN.

This YANG model supports both version 2 and version 3 of VRRP. VRRP version 2 defined in [RFC3768] supports IPv4. VRRP version 3 defined in [RFC5798] supports both IPv4 and IPv6.

1.1. Terminology

The keywords "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14, [RFC2119].

The following terms are defined in [RFC6020] and are not redefined here:

- augment

- data model
- data node

2. VRRP YANG model overview

This document defines the YANG module "ietf-vrrp", which has the following structure:

```
module: ietf-vrrp
augment /if:interfaces/if:interface/ip:ipv4:
   +--rw vrrp
      +--rw vrrp-instance* [vrid]
         +--rw vrid                              uint8
         +--rw version?                          enumeration
         +--rw log-state-change?                 boolean
         +--rw preempt!
         |  +--rw hold-time?   uint16
         +--rw priority?                         uint8
         +--rw accept-mode?                      boolean
         +--rw (advertise-interval-choice)?
         |  +--:(v2)
         |  |  +--rw advertise-interval-sec?         uint8
         |  +--:(v3)
         |     +--rw advertise-interval-centi-sec?   uint16
         +--rw track
         |  +--rw interfaces
         |  |  +--rw interface* [interface]
         |  |     +--rw interface             if:interface-ref
         |  |     +--rw priority-decrement?   uint8
         |  +--rw networks
         |     +--rw network* [network]
         |        +--rw network               inet:ipv4-prefix
         |        +--rw priority-decrement?   uint8
         +--rw virtual-ipv4-addresses
            +--rw virtual-ipv4-address* [ipv4-address]
               +--rw ipv4-address    inet:ipv4-address
augment /if:interfaces/if:interface/ip:ipv6:
   +--rw vrrp
      +--rw vrrp-instance* [vrid]
         +--rw vrid                              uint8
         +--rw version?                          enumeration
```

```
            +--rw log-state-change?               boolean
            +--rw preempt!
            |  +--rw hold-time?   uint16
            +--rw priority?                       uint8
            +--rw accept-mode?                    boolean
            +--rw advertise-interval-centi-sec?   uint16
            +--rw track
            |  +--rw interfaces
            |  |  +--rw interface* [interface]
            |  |     +--rw interface             if:interface-ref
            |  |     +--rw priority-decrement?   uint8
            |  +--rw networks
            |     +--rw network* [network]
            |        +--rw network               inet:ipv6-prefix
            |        +--rw priority-decrement?   uint8
            +--rw virtual-ipv6-addresses
               +--rw virtual-ipv6-address* [ipv6-address]
                  +--rw ipv6-address    inet:ipv6-address
   augment /if:interfaces-state/if:interface/ip:ipv4:
      +--ro vrrp
         +--ro vrrp-instance* [vrid]
            +--ro vrid                            uint8
            +--ro version?                        enumeration
            +--ro log-state-change?               boolean
            +--ro preempt!
            |  +--ro hold-time?   uint16
            +--ro priority?                       uint8
            +--ro accept-mode?                    boolean
            +--ro (advertise-interval-choice)?
            |  +--:(v2)
            |  |  +--ro advertise-interval-sec?         uint8
            |  +--:(v3)
            |     +--ro advertise-interval-centi-sec?   uint16
            +--ro track
            |  +--ro interfaces
            |  |  +--ro interface* [interface]
            |  |     +--ro interface             if:interface-ref
            |  |     +--ro priority-decrement?   uint8
            |  +--ro networks
            |     +--ro network* [network]
            |        +--ro network               inet:ipv4-prefix
```

```
            |       +--ro priority-decrement?   uint8
            +--ro virtual-ipv4-addresses
            |  +--ro virtual-ipv4-address* [ipv4-address]
            |     +--ro ipv4-address    inet:ipv4-address
            +--ro state?                          identityref
            +--ro is-owner?                       boolean
            +--ro last-adv-source?                inet:ip-address
            +--ro up-time?                        yang:date-and-time
            +--ro master-down-interval?           uint32
            +--ro skew-time?                      uint32
            +--ro last-event?                     string
            +--ro new-master-reason?              new-master-reason-type
            +--ro statistics
               +--ro discontinuity-time?        yang:date-and-time
               +--ro master-transitions?        yang:counter32
               +--ro advertisement-recv?        yang:counter64
               +--ro advertisement-sent?        yang:counter64
               +--ro interval-errors?           yang:counter64
   {validate-interval-errors}?
               +--ro priority-zero-pkts-rcvd?   yang:counter64
               +--ro priority-zero-pkts-sent?   yang:counter64
               +--ro invalid-type-pkts-rcvd?    yang:counter64
               +--ro address-list-errors?       yang:counter64
   {validate-address-list-errors}?
               +--ro packet-length-errors?      yang:counter64
   augment /if:interfaces-state/if:interface/ip:ipv6:
      +--ro vrrp
         +--ro vrrp-instance* [vrid]
            +--ro vrid                            uint8
            +--ro version?                        enumeration
            +--ro log-state-change?               boolean
            +--ro preempt!
            |  +--ro hold-time?   uint16
            +--ro priority?                       uint8
            +--ro accept-mode?                    boolean
            +--ro advertise-interval-centi-sec?   uint16
            +--ro track
            |  +--ro interfaces
            |  |  +--ro interface* [interface]
            |  |     +--ro interface             if:interface-ref
            |  |     +--ro priority-decrement?   uint8
```

```
            |  +--ro networks
            |     +--ro network* [network]
            |        +--ro network               inet:ipv6-prefix
            |        +--ro priority-decrement?   uint8
            +--ro virtual-ipv6-addresses
            |  +--ro virtual-ipv6-address* [ipv6-address]
            |     +--ro ipv6-address    inet:ipv6-address
            +--ro state?                          identityref
            +--ro is-owner?                       boolean
            +--ro last-adv-source?                inet:ip-address
            +--ro up-time?                        yang:date-and-time
            +--ro master-down-interval?           uint32
            +--ro skew-time?                      uint32
            +--ro last-event?                     string
            +--ro new-master-reason?              new-master-reason-type
            +--ro statistics
               +--ro discontinuity-time?        yang:date-and-time
               +--ro master-transitions?        yang:counter32
               +--ro advertisement-recv?        yang:counter64
               +--ro advertisement-sent?        yang:counter64
               +--ro interval-errors?           yang:counter64
   {validate-interval-errors}?
               +--ro priority-zero-pkts-rcvd?   yang:counter64
               +--ro priority-zero-pkts-sent?   yang:counter64
               +--ro invalid-type-pkts-rcvd?    yang:counter64
               +--ro address-list-errors?       yang:counter64
   {validate-address-list-errors}?
               +--ro packet-length-errors?      yang:counter64
   augment /if:interfaces-state:
      +--ro vrrp-global
         +--ro virtual-routers?                          uint32
         +--ro interfaces?                               uint32
         +--ro checksum-errors?                          yang:counter64
         +--ro version-errors?                           yang:counter64
         +--ro vrid-errors?                              yang:counter64
         +--ro ip-ttl-errors?                            yang:counter64
         +--ro global-statistics-discontinuity-time?   yang:date-and-
   time
   notifications:
      +---n vrrp-new-master-event
      |  +--ro master-ipaddr?       inet:ipv4-address
```

```
      |  +--ro new-master-reason?   new-master-reason-type
      +---n vrrp-protocol-error-event
      |  +--ro protocol-error-reason?   enumeration
      +---n vrrp-virtual-router-error-event
         +--ro interface?                    if:interface-ref
         +--ro ip-version?                   enumeration
         +--ro vrid-v4?                      leafref
         +--ro vrid-v6?                      leafref
         +--ro virtual-router-error-reason?  enumeration
```

3. VRRP YANG module

```
<CODE BEGINS> file "ietf-vrrp@2015-09-28.yang"
module ietf-vrrp {
  namespace "urn:ietf:params:xml:ns:yang:ietf-vrrp";
  // replace with IANA namespace when assigned
  prefix vrrp;

  import ietf-inet-types {
    prefix "inet";
  }

  import ietf-yang-types {
    prefix "yang";
  }

  import ietf-interfaces {
    prefix if;
  }

  import ietf-ip {
    prefix ip;
  }

  organization "TBD";
  contact "TBD";
  description
    "This YANG module defines a model for managing Virtual Router
    Redundancy Protocol (VRRP) version 2 and version 3.";

  revision "2015-09-28" {
```

```
      description "Initial revision";
      reference
        "RFC 2787: Definitions of Managed Objects for the Virtual
         Router Redundancy Protocol.
         RFC 3768: Virtual Router Redundancy Protocol (VRRP).
         RFC 5798: Virtual Router Redundancy Protocol (VRRP) Version
         3.
         RFC 6527: Definitions of Managed Objects for the Virtual
         Router Redundancy Protocol Version 3 (VRRPv3).";
    }

    /*
     * Features
     */

    feature validate-interval-errors {
      description
        "This feature indicates that the system validates that
        the advertisement interval from advertisement packets
        received is the same as the one configured for the local
        VRRP router.";
    }

    feature validate-address-list-errors {
      description
        "This feature indicates that the system validates that
        the address list from received packets matches the
        locally configured list for the VRRP router.";
    }

    /*
     * Typedefs
     */

    typedef new-master-reason-type {
      type enumeration {
        enum not-master {
          description
            "The virtual router has never transitioned to master
            state,";
        }
```

```
          enum priority {
            description "Priority was higher.";
          }
          enum preempted {
            description "The master was preempted.";
          }
          enum master-no-response {
            description "Previous master did not respond.";
          }
        }
        description
          "The reason for the virtual router to transition to master
          state.";
      } // new-master-reason-type

      /*
       * Identities
       */

      identity vrrp-state-type {
        description
          "The type to indicate the state of a virtual router.";
      }
      identity initialize {
        base vrrp-state-type;
        description
          "Indicates that the virtual router is waiting
          for a startup event.";
      }
      identity backup {
        base vrrp-state-type;
        description
          "Indicates that the virtual router is monitoring the
          availability of the master router.";
      }
      identity master {
        base vrrp-state-type;
        description
          "Indicates that the virtual router is forwarding
          packets for IP addresses that are associated with
          this virtual router.";
```

```
    }

    /*
     * Groupings
     */

    grouping vrrp-common-attributes {
      description
        "Group of VRRP attributes common to version 2 and version 3";

      leaf vrid {
        type uint8 {
          range 1..255;
        }
        description "Virtual router ID.";
      }

      leaf version {
        type enumeration {
          enum 2 {
            description "VRRP version 2.";
          }
          enum 3 {
            description "VRRP version 3.";
          }
        }
        description "Version 2 or version 3 of VRRP.";
      }

      leaf log-state-change {
        type boolean;
        description
          "Generates VRRP state change messages each time the VRRP
          instance changes state (from up to down or down to up).";
      }

      container preempt {
        presence "Present if preempt is enabled.";
        description
          "Enables a higher priority Virtual Router Redundancy
          Protocol (VRRP) backup router to preempt a lower priority
```

```
          VRRP master.";
        leaf hold-time {
          type uint16;
          description
            "Hold time, in seconds, for which a higher priority VRRP
            backup router must wait before preempting a lower priority
            VRRP master.";
        }
      }

      leaf priority {
        type uint8 {
          range 1..254;
        }
        default 100;
        description
          "Configures the Virtual Router Redundancy Protocol (VRRP)
          election priority for the backup virtual router.";
      }
    } // vrrp-common-attributes

    grouping vrrp-v3-attributes {
      description
        "Group of VRRP versin 3 attributes.";

      leaf accept-mode {
        type boolean;
        default false;
        description
          "Controls whether a virtual router in Master state will
          accept packets addressed to the address owner's IPvX address
          as its own if it is not the IPvX address owner. The default
          is false. Deployments that rely on, for example, pinging the
          address owner's IPvX address may wish to configure
          accept-mode to true.

          Note: IPv6 Neighbor Solicitations and Neighbor Advertisements
          MUST NOT be dropped when accept-mode is false.";
      }
    }
```

```
     grouping vrrp-ipv4-attributes {
       description
         "Group of VRRP attributes for IPv4.";

       uses vrrp-common-attributes;

       uses vrrp-v3-attributes {
         when "version = 3" {
           description "Applicable only to version 3.";
         }
       }

       choice advertise-interval-choice {
         description
           "The options for the advertisement interval at which VRRPv2
           or VRRPv3 advertisements are sent from the specified
           interface.";

         case v2 {
           when "version = 2" {
             description "Applicable only to version 2.";
           }
           leaf advertise-interval-sec {
             type uint8 {
               range 1..254;
             }
             default 1;
             description
               "Configures the interval that Virtual Router
               Redundancy Protocol Version 2 (VRRPv2) advertisements
               are sent from the specified interface.";
           }
         }

         case v3 {
           when "version = 3" {
             description "Applicable only to version 3.";
           }
           leaf advertise-interval-centi-sec {
             type uint16 {
               range 1..4095;
```

```
            }
            units centiseconds;
            default 100;
            description
              "Configures the interval that Virtual Router
              Redundancy Protocol version 3 (VRRPv3) advertisements
              are sent from the specified interface.";
          }
        }
      } // advertise-interval-choice

      container track {
        description
          "Enables the specified VRRP instance to track interfaces
          or networks.";
        container interfaces {
          description
            "Enables the specified Virtual Router Redundancy Protocol
            version 2 (VRRP) or version 3 (VRRPv3) instance to track
            an interface.";

          list interface {
            key "interface";
            description
              "Interface to track.";

            leaf interface {
              type if:interface-ref;
              must "../../../../../../../ipv4" {
                description "Interface is IPv4.";
              }
              description
                "Interface to track.";
            }

            leaf priority-decrement {
              type uint8 {
                range 1..254;
              }
              description
                "Specifies how much to decrement the priority of the
```

```
                VRRP instance if the interface goes down.";
            }
          } // track-interface
        } // track-interfaces

        container networks {
          description
            "Enables the backup Virtual Router Redundancy Protocol
            version 2 (VRRP) or version 3 (VRRPv3) router to track a
            specified network through the IP network prefix of that
            network.";
          list network {
            key "network";
            description
              "Enables the specified Virtual Router Redundancy
              Protocol version 2 (VRRP) or version 3 (VRRPv3)
              instance to track an interface.";

            leaf network {
              type inet:ipv4-prefix;
              description
                "Network to track.";
            }

            leaf priority-decrement {
              type uint8 {
                range 1..254;
              }
              default 10;
              description
                "Specifies how much to decrement the priority of the
                backup VRRP router if there is a failure in the IP
                network.";
            }
          } // track-network
        } // track-networks
      } // track

      container virtual-ipv4-addresses {
        description
          "Configures the virtual IP address for the Virtual Router
```

```
          Redundancy Protocol (VRRP) interface.";

        list virtual-ipv4-address {
          key "ipv4-address";
          max-elements 16;
          description
            "Virtual IP addresses for a single VRRP instance. For a
            VRRP owner router, the virtual address must match one
            of the IP addresses configured on the interface
            corresponding to the virtual router.";

          leaf ipv4-address {
            type inet:ipv4-address;
                description
                  "Virtual IPv4 address.";
          }
        } // virtual-ipv4-address
      } // virtual-ipv4-addresses
    } // grouping vrrp-ipv4-attributes

    grouping vrrp-ipv6-attributes {
      description
        "Group of VRRP attributes for IPv6.";

      uses vrrp-common-attributes;

      uses vrrp-v3-attributes {
        when "version = 3" {
          description "Uses VRRP version 3 attributes.";
        }
      } // uses vrrp-v3-attributes

      leaf advertise-interval-centi-sec {
        type uint16 {
          range 1..4095;
        }
        units centiseconds;
        default 100;
        description
          "Configures the interval that Virtual Router
          Redundancy Protocol version 3 (VRRPv3) advertisements
```

```
          are sent from the specified interface.";
        }

        container track {
          description
            "Enables the specified VRRP instance to track interfaces
            or networks.";
          container interfaces {
            description
              "Enables the specified Virtual Router Redundancy Protocol
              version 2 (VRRP) or version 3 (VRRPv3) instance to track
              an interface.";
            list interface {
              key "interface";
              description
                "Interface to track.";

              leaf interface {
                type if:interface-ref;
                must "../../../../../../../ipv6" {
                  description "Interface is IPv6.";
                }
                description
                  "Interface to track.";
              }

              leaf priority-decrement {
                type uint8 {
                  range 1..254;
                }
                description
                  "Specifies how much to decrement the priority of the
                  VRRP instance if the interface goes down.";
              }
            } // track-interface
          } // track-interfaces

          container networks {
            description
              "Enables the backup Virtual Router Redundancy Protocol
              version 2 (VRRP) or version 3 (VRRPv3) router to track a
```

```
            specified network through the IP network prefix of that
            network.";
          list network {
            key "network";
            description
              "Enables the specified Virtual Router Redundancy
              Protocol version 2 (VRRP) or version 3 (VRRPv3)
              instance to track an interface.";

            leaf network {
              type inet:ipv6-prefix;
              description
                "Network to track.";
            }

            leaf priority-decrement {
              type uint8 {
                range 1..254;
              }
              default 10;
              description
                "Specifies how much to decrement the priority of the
                backup VRRP router if there is a failure in the IP
                network.";
            }
          } // track-network
        } // track-networks
      } // track

      container virtual-ipv6-addresses {
        description
          "Configures the virtual IP address for the Virtual Router
           Redundancy Protocol (VRRP) interface.";
        list virtual-ipv6-address {
          key "ipv6-address";
          max-elements 2;
          description
            "Two IPv6 addresses are allowed. The first one must be
            a link-local address and the second one can be a
            link-local or global address.";
```

```
            leaf ipv6-address {
              type inet:ipv6-address;
              description
                "Virtual IPv6 address.";
            }
          } // virtual-ipv6-address
        } // virtual-ipv6-addresses
      } // grouping vrrp-ipv6-attributes

      grouping vrrp-state-attributes {
        description
          "Group of VRRP state attributes.";

        leaf state {
          type identityref {
            base vrrp-state-type;
          }
          description
            "Operational state.";
        }

        leaf is-owner {
          type boolean;
          description
            "Set to true if this virtual router is owner.";
        }

        leaf last-adv-source {
          type inet:ip-address;
          description
            "Last advertised IPv4/IPv6 source address";
        }

        leaf up-time {
          type yang:date-and-time;
          description
            "The time when this virtual router
            transitioned out of init state.";
        }

        leaf master-down-interval {
```

```
          type uint32;
          units centiseconds;
          description
            "Time interval for backup virtual router to declare
            Master down.";
        }

        leaf skew-time {
          type uint32;
          units microseconds;
          description
            "Calculated based on the priority and advertisement
            interval configuration command parameters. See RFC 3768.";
        }

        leaf last-event {
          type string;
          description
            "Last reported event.";
        }

        leaf new-master-reason {
          type new-master-reason-type;
          description
            "Indicates the reason for the virtual router to transition
            to master state.";
        }

        container statistics {
          description
            "VRRP statistics.";

          leaf discontinuity-time {
            type yang:date-and-time;
            description
              "The time on the most recent occasion at which any one or
              more of the VRRP statistic counters suffered a
              discontinuity.  If no such discontinuities have occurred
              since the last re-initialization of the local management
              subsystem, then this node contains the time that the
              local management subsystem re-initialized itself.";
```

```
        }

        leaf master-transitions {
          type yang:counter32;
          description
            "The total number of times that this virtual router's
            state has transitioned to master";
        }

        leaf advertisement-recv {
          type yang:counter64;
          description
            "The total number of VRRP advertisements received by
             this virtual router.";
        }

        leaf advertisement-sent {
          type yang:counter64;
          description
            "The total number of VRRP advertisements sent by
            this virtual router.";
        }

        leaf interval-errors {
          if-feature validate-interval-errors;
          type yang:counter64;
          description
            "The total number of VRRP advertisement packets
            received with an advertisement interval
            different than the one configured for the local
            virtual router";
        }

        leaf priority-zero-pkts-rcvd {
          type yang:counter64;
          description
            "The total number of VRRP packets received by the
            virtual router with a priority of 0.";
        }

        leaf priority-zero-pkts-sent {
```

```
          type yang:counter64;
          description
            "The total number of VRRP packets sent by the
            virtual router with a priority of 0.";
        }

        leaf invalid-type-pkts-rcvd {
          type yang:counter64;
          description
            "The number of VRRP packets received by the virtual
            router with an invalid value in the 'type' field.";
        }

        leaf address-list-errors {
          if-feature validate-address-list-errors;
          type yang:counter64;
          description
            "The total number of packets received with an
            address list that does not match the locally
            configured address list for the virtual router.";
        }

        leaf packet-length-errors {
          type yang:counter64;
          description
            "The total number of packets received with a packet
            length less than the length of the VRRP header.";
        }
      } // container statistics
    } // grouping vrrp-state-attributes

    grouping vrrp-global-state-attributes {
      description
        "Group of VRRP global state attributes.";

      leaf virtual-routers {
        type uint32;
        description "Number of configured virtual routers.";
      }

      leaf interfaces {
```

```
          type uint32;
          description "Number of interface with VRRP configured.";
        }

        leaf checksum-errors {
          type yang:counter64;
          description
            "The total number of VRRP packets received with an invalid
            VRRP checksum value.";
          reference "RFC 5798, Section 5.2.8";
        }

        leaf version-errors {
          type yang:counter64;
          description
            "The total number of VRRP packets received with an unknown
            or unsupported version number.";
          reference "RFC 5798, Section 5.2.1";
        }

        leaf vrid-errors {
          type yang:counter64;
          description
            "The total number of VRRP packets received with a VRID that
            is not valid for any virtual router on this router.";
          reference "RFC 5798, Section 5.2.3";
        }

        leaf ip-ttl-errors {
          type yang:counter64;
          description
            "The total number of VRRP packets received by the
            virtual router with IP TTL (Time-To-Live) not equal
            to 255.";
          reference "RFC 5798, Sections 5.1.1.3 and 5.1.2.3.";
        }

        leaf global-statistics-discontinuity-time {
            type yang:date-and-time;
            description
              "The time on the most recent occasion at which one of
```

```
            router-checksum-errors, router-version-errors,
            router-vrid-errors, and ip-ttl-errors suffered a
            discontinuity.

            If no such discontinuities have occurred since the last
            re-initialization of the local management subsystem,
            then this object will be 0.";
      }
    } // vrrp-global-state-attributes

    /*
     * Configuration data nodes
     */

    augment "/if:interfaces/if:interface/ip:ipv4" {
      description "Augment IPv4 interface.";

      container vrrp {
        description
          "Configures the Virtual Router Redundancy Protocol (VRRP)
          version 2 or version 3 for IPv4.";

        list vrrp-instance {
          key vrid;
          description
            "Defines a virtual router, identified by a virtual router
            identifier (VRID), within IPv4 address space.";

          uses vrrp-ipv4-attributes;
        }
      }
    } // augment ipv4

    augment "/if:interfaces/if:interface/ip:ipv6" {
      description "Augment IPv6 interface.";

      container vrrp {
        description
          "Configures the Virtual Router Redundancy Protocol (VRRP)
          version 3 for IPv6.";
```

```
        list vrrp-instance {
          must "version = 3" {
            description
              "IPv6 is only supported by version 3.";
          }
          key vrid;
          description
            "Defines a virtual router, identified by a virtual router
            identifier (VRID), within IPv6 address space.";

          uses vrrp-ipv6-attributes;
        } // list vrrp-instance
      } // container vrrp
    } // augment ipv6

    /*
     * Operational state data nodes
     */

    augment "/if:interfaces-state/if:interface/ip:ipv4" {
      description "Augment IPv4 interface state.";

      container vrrp {
        description
          "State information for Virtual Router Redundancy Protocol
          (VRRP) version 2 for IPv4.";

        list vrrp-instance {
          key vrid;
          description
            "States of a virtual router, identified by a virtual router
            identifier (VRID), within IPv4 address space.";

          uses vrrp-ipv4-attributes;
          uses vrrp-state-attributes;
        } // list vrrp-instance
      }
    }

    augment "/if:interfaces-state/if:interface/ip:ipv6" {
      description "Augment IPv6 interface state.";
```

```
      container vrrp {
        description
          "State information of the Virtual Router Redundancy Protocol
          (VRRP) version 2 or version 3 for IPv6.";

        list vrrp-instance {
          key vrid;
          description
            "States of a virtual router, identified by a virtual router
            identifier (VRID), within IPv6 address space.";

          uses vrrp-ipv6-attributes;
          uses vrrp-state-attributes;
        } // list vrrp-instance
      }
    }

    augment "/if:interfaces-state" {
      description "Specify VRRP state data at the global level.";

      container vrrp-global {
        description
          "State information of the Virtual Router Redundancy Protocol
          (VRRP) at the global level";

        uses vrrp-global-state-attributes;
      }
    }

    /*
     * Notifications
     */

    notification vrrp-new-master-event {
      description
        "Notification event for a change of VRRP new master.";
      leaf master-ipaddr {
        type inet:ipv4-address;
        description
          "IPv4 or IPv6 address of the new master.";
```

```
      }
      leaf new-master-reason {
        type new-master-reason-type;
        description
          "Indicates the reason for the virtual router to transition
          to master state.";
      }
    }

    notification vrrp-protocol-error-event {
      description
        "Notification event for a VRRP protocol error.";
      leaf protocol-error-reason {
        type enumeration {
          enum checksum-error {
            description
              "A packet has been received with an invalid VRRP checksum
              value.";
          }
          enum version-error {
            description
              "A packet has been received with an unknown or
              unsupported version number.";
          }
          enum vrid-error {
            description
              "A packet has been received with a VRID that is not valid
              for any virtual router on this router.";
          }
          enum ip-ttl-error {
            description
              "A packet has been received with IP TTL (Time-To-Live)
              not equal to 255.";
          }
        }
        description
          "Indicates the reason for the protocol error.";
      }
    }

    notification vrrp-virtual-router-error-event {
```

```
      description
        "Notification event for a error happened on a virtual router.";
      leaf interface {
        type if:interface-ref;
        description
          "Indicates the interface for which statistics area
          to be cleared.";
      }
      leaf ip-version {
        type enumeration {
          enum 4 {
            description "IPv4";
          }
          enum 6 {
            description "IPv6";
          }
        }
        description "Indicates the IP version.";
      }
      leaf vrid-v4 {
        type leafref {
          path "/if:interfaces/if:interface"
            + "[if:name = current()/../interface]/ip:ipv4/vrrp/"
            + "vrrp-instance/vrid";
        }
        description
          "Indicates the virtual router on which the event has
          occured.";
      }

      leaf vrid-v6 {
        type leafref {
          path "/if:interfaces/if:interface"
            + "[if:name = current()/../interface]/ip:ipv6/vrrp/"
            + "vrrp-instance/vrid";
        }
        description
          "Indicates the virtual router on which the event has
          occured.";
        }
      leaf virtual-router-error-reason {
```

```
        type enumeration {
          enum interval-error {
            description
              "A packet has been received with an advertisement
              interval different than the one configured for the local
              virtual router";
          }
          enum address-list-error {
            description
              "A packet has been received with an address list that
              does not match the locally configured address list for
              the virtual router.";
          }
          enum packet-length-error {
            description
              "A packet has been received with a packet length less
              than the length of the VRRP header.";
          }
        }
        description
          "Indicates the reason for the virtual router error.";
      }
    }
  }
  <CODE ENDS>
```

4. Security Considerations

The configuration, state, action and notification data defined in this document are designed to be accessed via the NETCONF protocol [RFC6241]. The data-model by itself does not create any security implications. The security considerations for the NETCONF protocol are applicable. The NETCONF protocol used for sending the data supports authentication and encryption.

5. Contributors

Yuyang Xie
Huawei Technologies
No. 156 Beiqing Rd. Haidian District
Beijing 100095
P.R. China

Email: xieyuyang@huawei.com

6. References

6.1. Normative References

[RFC6020] Bjorklund, M., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, October 2010.

[RFC6021] Schoenwaelder, J., "Common YANG Data Types", RFC 6021, October 2010.

[RFC6241] Enns, R., Bjorklund, M., Schoenwaelder, J., and A. Bierman, "Network Configuration Protocol (NETCONF)", RFC 6241, June 2011.

[RFC2234] Crocker, D. and Overell, P.(Editors), "Augmented BNF for Syntax Specifications: ABNF", RFC 2234, Internet Mail Consortium and Demon Internet Ltd., November 1997.

[RFC2338] Knight, S., Weaver, D., Whipple, D., Hinden, R., Mitzel, D., Hunt, P., Higginson, P., Shand, M., and A. Lindem, "Virtual Router Redundancy Protocol", RFC 2338, April 1998.

[RFC2787] Jewell, B. and D. Chuang, "Definitions of Managed Objects for the Virtual Router Redundancy Protocol", RFC 2787, March 2000.

[RFC5798] Nadas, S., Ed., "Virtual Router Redundancy Protocol (VRRP) Version 3 for IPv4 and IPv6", RFC 5798, March 2010.

[RFC6527] Tata, K., Ed., "Definitions of Managed Objects for the Virtual Router Redundancy Protocol Version 3 (VRRPv3)", RFC 6527, March 2012.

6.2. Informative References

[RFC6087] Bierman, A., "Guidelines for Authors and Reviewers of YANG Data Model Documents", RFC 6087, January 2011.

Authors' Addresses

Xufeng Liu (Editor)
Ericsson
1595 Spring Hill Road, Suite 500
Vienna, VA 22182
USA

Email: xliu@kuatrotech.com

Athanasios Kyparlis
Ericsson
1595 Spring Hill Road, Suite 500
Vienna, VA 22182
USA

Email: athanasios.kyparlis@ericsson.com

Ravi Parikh
Ericsson
300 Holger Way
San Jose, CA 95134
USA

Email: ravi.parikh@ericsson.com

Acee Lindem
Cisco Systems
301 Midenhall Way
Cary, NC 27513
USA

Email: acee@cisco.com

Mingui Zhang
Huawei Technologies
No. 156 Beiqing Rd. Haidian District
Beijing 100095
P.R. China

Email: zhangmingui@huawei.com

draft-mwnpkazcap-rtgwg-common-oam-00

RTG Working Group
Internet-Draft
Intended status: Informational
Expires: April 21, 2016

G. Mirsky
R. White
Ericsson
E. Nordmark
Arista Networks
C. Pignataro
N. Kumar
Cisco Systems, Inc.
S. Aldrin
Google
L. Zheng
M. Chen
Huawei Technologies
N. Akiya
Big Switch Networks
S. Pallagatti
Juniper Networks
October 19, 2015

Rationale for Transport-independent Common Operations, Administration and Maintenance (OAM) draft-mwnpkazcap-rtgwg-common-oam-00

Abstract

This document discusses set of Operations, Administration and Maintenance (OAM) tools that can be used as common OAM independent of specific encapsulation at server layer. Requirements toward server layer to support common OAM are listed as well.

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, 2016.

Table of Contents

1. Introduction

The introduction and development of new service layers such as Service Function Chaining (SFC) and Bit-Ingress Explicit Replication (BIER), is driving the need for new Operations, Administration and Maintenance (OAM) tools. This document discusses benefits of Common transport independent OAM solution to support components of network management framework known as Fault, Configuration, Accounting, Performance, and Security (FCAPS):

- Fault monitoring and defect localization;

- Performance measurement, both passive and active.

1.1. Conventions used in this document

1.1.1. Terminology

The term "OAM" used in this document interchangeably with longer version "set of OAM protocols, methods and tools for a particular layer".

BIER: Bit-Ingress Explicit Replication

FCAPS: Fault, Configuration, Accounting, Performance, and Security

OAM: Operations, Administration and Maintenance

SFC: Service Function Chaining

1.1.2. Requirements Language

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

2. Use Case for Common OAM

Recently several new service layers have been developed in IETF. Each of responsible groups, e.g. SPRING, NVO3, SFC, BIER, have formulated a set of OAM requirements, specific to their respective layer [I-D.ietf-spring-sr-oam-requirement], [I-D.ashwood-nvo3-oam-requirements], [I-D.ietf-sfc-oam-framework], and [I-D.ietf-bier-oam-requirements]. Proposals have already been put forward to satisfy those requirements, though mostly by enhancing existing OAM tools, such as LSP Ping [I-D.kumarkini-mpls-spring-lsp-ping]. Enhancing existing tools certainly leads to faster deployment of OAM but may create operational issues later on. For instance, these new service layers may be implemented a wide range of transport layers, e.g. MPLS or IPv6, so OAM tools that are transport-oriented like LSP Ping would not be able to perform end-to-end for inter-domain scenario.

At the same time, the Bidirectional Forwarding Detection (BFD) protocol is being successfully adopted for IPv6 and MPLS networks, and efforts are moving forward to define transport-independent OAM tool based only on the requirements of one of these new services, BIER.

[I-D.ietf-rtgwg-dt-encap] raised question of common OAM for NVO3, SFC, and BIER. We want to take this further and:

- o analyze relevant OAM requirements and document common set of requirements towards OAM as well as requirements toward aservice layer to enable its ability to support OAM;

- o analyze OAM solutions (proactive and on-demand CC/CV, PM, FM) being proposed and formulate approach to structure OAM tools that may be re-used across several types on encapsulation.

3. IANA Considerations

This document does not propose any IANA consideration. This section may be removed.

4. Security Considerations

This document does not raise any security concerns or issues in addition to ones common to networking.

5. Acknowledgement

TBD

6. References

6.1. Normative References

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

6.2. Informative References

[I-D.ashwood-nvo3-oam-requirements] Chen, H., Ashwood-Smith, P., Xia, L., Iyengar, R., Tsou, T., Sajassi, A., Boucadair, M., Jacquenet, C., Daikoku, M., Ghanwani, A., and R. Krishnan, "NVO3 Operations, Administration, and Maintenance Requirements", draft-ashwood-nvo3-oam-requirements-04 (work in progress), October 2015.

[I-D.ietf-bier-oam-requirements] Mirsky, G., Nordmark, E., Pignataro, C., Kumar, N., Aldrin, S., Zheng, L., Chen, M., Akiya, N., and J. Networks, "Operations, Administration and Maintenance (OAM) Requirements for Bit Index Explicit Replication (BIER) Layer", draft-ietf-bier-oam-requirements-00 (work in progress), September 2015.

[I-D.ietf-rtgwg-dt-encap]
Nordmark, E., Tian, A., Gross, J., Hudson, J., Kreeger, L., Garg, P., Thaler, P., and T. Herbert, "Encapsulation Considerations", draft-ietf-rtgwg-dt-encap-00 (work in progress), July 2015.

[I-D.ietf-sfc-oam-framework]
Aldrin, S., Krishnan, R., Akiya, N., Pignataro, C., and A. Ghanwani, "Service Function Chaining Operation, Administration and Maintenance Framework", draft-ietf-sfc-oam-framework-00 (work in progress), August 2015.

[I-D.ietf-spring-sr-oam-requirement]
Kumar, N., Pignataro, C., Akiya, N., Geib, R., Mirsky, G., and S. Litkowski, "OAM Requirements for Segment Routing Network", draft-ietf-spring-sr-oam-requirement-00 (work in progress), June 2015.

[I-D.kumarkini-mpls-spring-lsp-ping]
Kumar, N., Swallow, G., Pignataro, C., Akiya, N., Kini, S., Gredler, H., and M. Chen, "Label Switched Path (LSP) Ping/Trace for Segment Routing Networks Using MPLS Dataplane", draft-kumarkini-mpls-spring-lsp-ping-04 (work in progress), July 2015.

Authors' Addresses

Greg Mirsky
Ericsson

Email: gregory.mirsky@ericsson.com

Russ White
Ericsson

Email: russ@riw.us

Erik Nordmark
Arista Networks

Email: nordmark@acm.org

Carlos Pignataro
Cisco Systems, Inc.

Email: cpignata@cisco.com

Nagendra Kumar
Cisco Systems, Inc.

Email: naikumar@cisco.com

Sam Aldrin
Google

Email: aldrin.ietf@gmail.com

Lianshu Zheng
Huawei Technologies

Email: vero.zheng@huawei.com

Mach Chen
Huawei Technologies

Email: mach.chen@huawei.com

Nobo Akiya
Big Switch Networks

Email: nobo.akiya.dev@gmail.com

Santosh Pallagatti
Juniper Networks

Email: santoshpk@juniper.net

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: February 11, 2017

N. Gupta
A. Dogra
Cisco Systems, Inc.
C. Docherty

G. Mirsky
Ericsson
J. Tantsura
Individual

August 10, 2016

Fast failure detection in VRRP with BFD
draft-nitish-vrrp-bfd-04

Abstract

This document describes how Bidirectional Forwarding Detection (BFD) can be used to support sub-second detection of a Master Router failure in the Virtual Router Redundancy Protocol (VRRP).

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 11, 2017.

Copyright Notice

Table of Contents

1. Introduction

The Virtual Router Redundancy Protocol (VRRP) provides redundant Virtual gateways in the Local Area Network (LAN), which is typically the first point of failure for end-hosts sending traffic out of the LAN. Fast failure detection of VRRP Master is critical in supporting high availability of services and improved Quality of Experience to users. In VRRP [RFC5798] specification, Backup routers depend on VRRP packets generated at a regular interval by the Master router, to detect the health of the VRRP Master. Faster failure detection can be achieved within VRRP protocol by reducing the Advertisement Interval and hold down timers. However, aggressive timers can put extra load on CPU and the network bandwidth which may not be desirable.

Since the VRRP protocol depends on the availability of Layer 3 IPv4 or IPv6 connectivity between redundant peers, the VRRP protocol can interact with the Layer 3 variant of BFD as described in [RFC5881] or [I-D.draft-ietf-bfd-multipoint] to achieve a much faster failure detection of the VRRP Master on the LAN. BFD, as specified by the [RFC5880] or [I-D.draft-ietf-bfd-multipoint] can provide a much faster failure detection in the range of 150ms, if implemented in the part of a Network device which scales better than VRRP when aggressive timers are used.

2. Requirements Language

In this document, several words are used to signify the requirements of the specification. 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]

3. Applicability of Single-hop BFD

BFD for IPv4 or IPv6 (Single Hop) [RFC5881] requires that in order for a BFD session to be formed both peers participating in a BFD session need to know its peer IPv4 or IPV6 address. This poses a unique problem with the definition of the VRRP protocol, that makes the use of BFD for IPv4 or IPv6 [RFC5881] more challenging. In VRRP it is only the Master router that sends Advert packets. This means that a Master router is not aware of any Backup routers, and Backup routers are only aware of the Master router. This also means that a Backup router is not aware of any other Backup routers in the Network.

Since BFD for IPv4 or IPv6 [RFC5881] requires that a session be formed by both peers using a full destination and source address, there needs to be some external means to provide this information to BFD on behalf of VRRP. Once the peer information is made available, VRRP can form BFD sessions with its peer Virtual Router. The BFD session for a given Virtual Router is identified as the Critical Path BFD Session, which is the session that forms between the current VRRP Master router, and the highest priority Backup router. When the Critical Path BFD Session identified by VRRP as having changed state from Up to Down, then this will be interpreted by the VRRP state machine on the highest priority Backup router as a Master Down event. A Master Down event means that the highest priority Backup peer will immediately become the new Master for the Virtual Router.

NOTE: At all times, the normal fail-over mechanism defined in the VRRP [RFC5798] will be unaffected, and the BFD fail-over mechanism will always resort to normal VRRP fail-over.

This draft defines the mechanism used by the VRRP protocol to build a peer table that will help in forming of BFD session and the detection of Critical Path BFD session. If the Critical Path BFD session were to go down, it will signal a Master Down event and make the most preferred Backup router as the VRRP Master router. This requires an extension to the VRRP protocol.

This can be achieved by defining a new type in the VRRP Advert packet, and allowing VRRP peers to build a peer table in any of the operational state, Master or Backup.

3.1. Extension to VRRP protocol

In this mode of operation VRRP peers learn the adjacent routers, and form BFD session between the learnt routers. In order to build the peer table, all routers send VRRP Advert packets whilst in any of the operational states (Master or Backup). Normally VRRP peers only send Advert packets whilst in the Master state, however in this mode VRRP Backup peers will also send Advert packets with the type field set to BACKUP ADVERTISEMENT type defined in Section 3.3 of this document. The VRRP Master router will still continue to send packets with the Advert type as ADVERTISEMENT as defined in the VRRP protocol. This is to maintain inter-operability with peers complying to VRRP protocol.

Additionally, Advert packets sent from Backup Peers must not use the Virtual router MAC address as the source address. Instead it must use the Interface MAC address as the source address from which the packet is sent from. This is because the source MAC override feature is used by the Master to send Advert packets from the Virtual Router MAC address, which is used to keep the bridging cache on LAN switches and bridging devices refreshed with the destination port for the Virtual Router MAC.

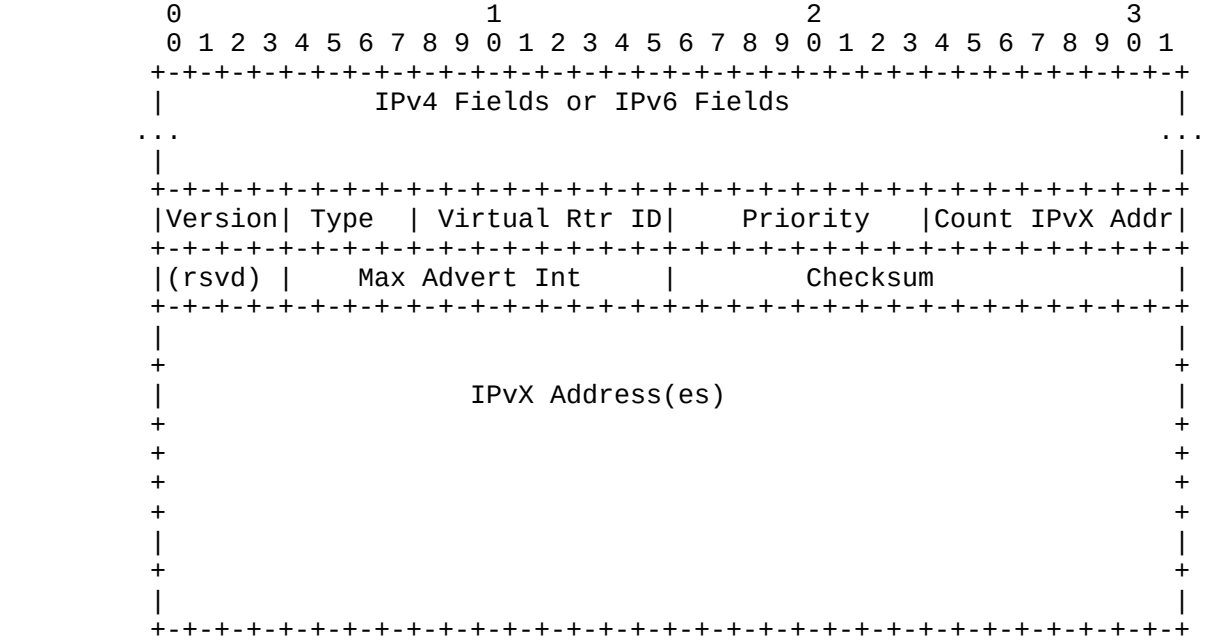
3.2. VRRP Peer Table

VRRP peers can now form the peer table by learning the source address in the ADVERTISEMENT or BACKUP ADVERTISEMENT packet sent by VRRP Master or Backup peers. This allows peers to create BFD sessions with other operational peers.

A peer entry should be removed from the peer table if Advert is not received from a peer for a period of (3 * the Advert interval).

3.3. VRRP BACKUP ADVERTISEMENT Packet Type

The following figure shows the VRRP packet as defined in VRRP [RFC5798] RFC.



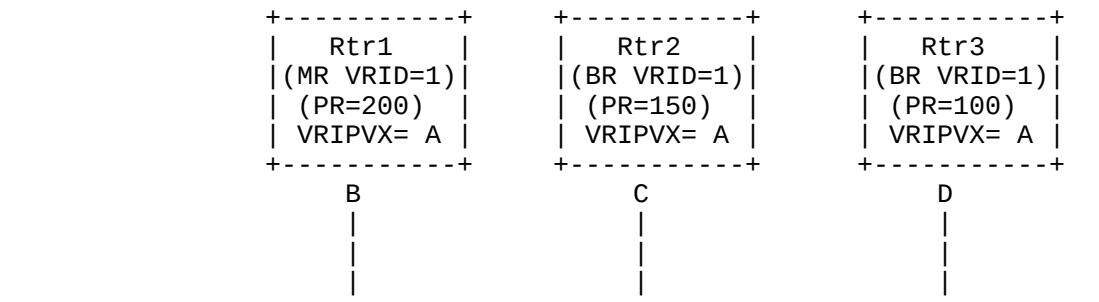
The type field specifies the type of this VRRP packet. The type field can have two values. Type 1 (ADVERTISEMENT) is used by the VRRP Master Router. Type 2 (BACKUP ADVERTISEMENT) is used by the VRRP Backup router. This is to distinguish the packets sent by the VRRP backup Router. VRRP Backup fills Backup_Advertisement_Interval in the Max Advert Int of BACKUP ADVERTISEMENT packet. Rest of the fields in Advert packet remain the same.

1 ADVERTISEMENT
2 BACKUP ADVERTISEMENT

A packet with unknown type MUST be discarded.

3.4. Sample configuration

The following figure shows a simple network with three VRRP routers implementing one virtual router.



Legend:

```
        ---+---+---+--    =  Ethernet, Token Ring, or FDDI
                      MR    =  Master Router
                      BR    =  Backup Router
                      PR    =  VRRP Router priority
                      VRID  =  VRRP Router ID
                      VRIPVX=  IPv4 or IPv6 address protected by
                               the VRRP Router
                      B,C,D =  Interface IPv4 or IPv6 address of
                               the Virtual Router
```

In the above configuration there are three routers on the LAN protecting an IPv4 or IPv6 address associated to a Virtual Router ID 1. Rtr1 is the Master router since it has the highest priority compared to Rtr2 and Rtr3. Now if peer learning extension is enabled on all the peers. Rtr1 will send the Advert packet with type field set to 1. While Rtr2 and Rtr3 will send the Advert packet with type field set to 2. In the above configuration the peer table built at each router is shown below:

Rtr1 Peer table

Peer Address	Priority
C	150
D	100

Rtr2 Peer table

Peer Address	Priority
B	200
D	100

Rtr3 Peer table

Peer Address	Priority
B	200
C	150

Once the peer tables are formed, VRRP on each router can form a BFD sessions with the learnt peers.

3.5. Critical BFD session

The Critical BFD Session is determined to be the session between the VRRP Master and the next best VRRP Backup. Failure of the Critical BFD session indicates that the Master is no longer available and the most preferred Backup will now become Master.

In the above example the Critical BFD session is shared between Rtr1 and Rtr2. If the BFD Session goes from Up to Down state, Rtr2 can treat it as a Master down event and immediately assume the role of VRRP Master router for VRID 1 and Rtr3 will become the critical Backup. If the priorities of two Backup routers are same then the primary IPvX Address of the sender is used to determine the highest priority Backup. Where higher IPvX address has higher priority.

3.6. Protocol State Machine

3.6.1. Parameters Per Virtual Router

Following parameters are added to the VRRP protocol to support this mode of operation.

Backup_Advertisement_Interval	Time interval between BACKUP ADVERTISEMENTS (centiseconds). Default is 100 centiseconds (1 second).
Backup_Adver_Interval	Advertisement interval contained in BACKUP ADVERTISEMENTS received from the Backup (centiseconds). This value is saved by virtual routers used, to compute Backup_Down_Interval.
Backup_Down_Interval	Time interval for VRRP instance to declare Backup down (centiseconds). Calculated as (3 * Backup_Adver_Interval) for each VRRP Backup.
Critical_Backup	Procedure outlined in section 3.4 of this document is used to determine the Critical_Backup at each VRRP Instance.
Critical_BFD_Session	The Critical BFD Session is the session between the VRRP Master and Critical_Backup.

3.6.2. Timers

Following timers are added to the VRRP protocol to support this mode of operation.

Backup_Down_Timer	Timer that fires when BACKUP ADVERTISEMENT has not been heard from a backup peer for Backup_Down_Interval.
Backup_Adver_Timer	Timer that fires to trigger sending of BACKUP ADVERTISEMENT based on Backup_Advertisement_Interval.

3.6.3. VRRP State Machine with BFD

Following State Machine replaces the state Machine outlined in section 6.4 of the VRRP protocol [RFC5798] to support this mode of operation. Please refer to the section 6.4 of [RFC5798] for State description.

3.6.3.1. Initialize

Following state machine replaces the state machine outlined in section 6.4.1 of [RFC5798]

(100) If a Startup event is received, then:

(105) - If the Priority = 255 (i.e., the router owns the IPvX address associated with the virtual router), then:

(110) + Send an ADVERTISEMENT

(115) + If the protected IPvX address is an IPv4 address, then:

(120) * Broadcast a gratuitous ARP request containing the virtual router MAC address for each IP address associated with the virtual router.

(125) + else // IPv6

(130) * For each IPv6 address associated with the virtual router, send an unsolicited ND Neighbor Advertisement with the Router Flag (R) set, the Solicited Flag (S) unset, the Override flag (O) set, the target address set to the IPv6 address of the virtual router, and the target link-layer address set to the virtual router MAC address.

(135) +endif // was protected addr IPv4?

(140) + Set the Adver_Timer to Advertisement_Interval

(145) + Transition to the {Master} state

(150) - else // rtr does not own virt addr

(155) + Set Master_Adver_Interval to Advertisement_Interval

(160) + Set the Master_Down_Timer to Master_Down_Interval

(165) + Set Backup_Adver_Timer to Backup_Advertisement_Interval

(170) + Transition to the {Backup} state

(175) -endif // priority was not 255

(180) endif // startup event was recv

3.6.3.2. Backup

Following state machine replaces the state machine outlined in section 6.4.2 of [RFC5798]

(300) While in this state, a VRRP router MUST do the following:

(305) - If the protected IPvX address is an IPv4 address, then:

(310) + MUST NOT respond to ARP requests for the IPv4 address(es) associated with the virtual router.

(315) - else // protected addr is IPv6

(320) + MUST NOT respond to ND Neighbor Solicitation messages for the IPv6 address(es) associated with the virtual router.

(325) + MUST NOT send ND Router Advertisement messages for the virtual router.

(330) -endif // was protected addr IPv4?

(335) - MUST discard packets with a destination link-layer MAC address equal to the virtual router MAC address.

(340) - MUST NOT accept packets addressed to the IPvX address(es) associated with the virtual router.

(345) - If a Shutdown event is received, then:

(350) + Cancel the Master_Down_Timer.

(355) + Cancel the Backup_Adver_Timer.

(360) + Cancel Backup_Down_Timers.

(365) + Remove Peer table.

(370) + If Critical_BFD_Session Exists:

(375) * Tear down the Critical_BFD_Session.

(380) + endif // Critical_BFD_Session Exists?

(385) + Send a BACKUP ADVERTISEMENT with Priority = 0.

(390) + Transition to the {Initialize} state.

```
(395) -endif // shutdown recv

(400) - If the Master_Down_Timer fires or
        If Critical_BFD_Session transitions from UP to DOWN, then:

   (405) + Send an ADVERTISEMENT

   (415) + If the protected IPvX address is an IPv4 address, then:

      (420) * Broadcast a gratuitous ARP request on that interface
      containing the virtual router MAC address for each IPv4
      address associated with the virtual router.

   (425) + else // ipv6

      (430) * Compute and join the Solicited-Node multicast
      address [RFC4291] for the IPv6 address(es) associated with
      the virtual router.

      (435) * For each IPv6 address associated with the virtual
      router, send an unsolicited ND Neighbor Advertisement with
      the Router Flag (R) set, the Solicited Flag (S) unset, the
      Override flag (O) set, the target address set to the IPv6
      address of the virtual router, and the target link-layer
      address set to the virtual router MAC address.

   (440) +endif // was protected addr ipv4?

   (445) + Set the Adver_Timer to Advertisement_Interval.

   (450) + If the Critical_BFD_Session exists:

      (455) @ Tear Critical_BFD_Session.

    (460) + endif // Critical_BFD_Session exists

   (465) + Calculate the Critical_Backup.

   (470) + If the Critical_Backup exists:

      (475) * BootStrap Critical_BFD_Session with the
              Critical_Backup.

   (480) + endif //Critical_Backup exists?

   (485) + Transition to the {Master} state.

(490) -endif // Master_Down_Timer fired
```

```
(485) - If an ADVERTISEMENT is received, then:

   (490) + If the Priority in the ADVERTISEMENT is zero, then:

      (495) * Set the Master_Down_Timer to Skew_Time.

      (500) * If the Critical_BFD_Session exists:

         (505) * Tear Critical_BFD_Session with the Master.

      (510) * endIf // Critical_BFD_Session exists

   (515) + else // priority non-zero

      (520) * If Preempt_Mode is False, or if the Priority in the
      ADVERTISEMENT is greater than or equal to the local
      Priority, then:

         (525) @ Set Master_Adver_Interval to Adver Interval
         contained in the ADVERTISEMENT.

         (530) @ Recompute the Master_Down_Interval.

         (535) @ Reset the Master_Down_Timer to
         Master_Down_Interval.

         (540) @ Determine Critical_Backup.

         (545) @ If Critical_BFD_Session does not exists and this
         instance is the Critical_Backup:

            (550) @+ BootStrap Critical_BFD_Session with Master.

         (555) @ endif //Critical_BFD_Session exists check

      (560) * else // preempt was true or priority was less

         (565) @ Discard the ADVERTISEMENT.

      (570) *endif // preempt test

   (575) +endif // was priority zero?

(580) -endif // was advertisement recv?

(585) - If a BACKUP ADVERTISEMENT is received, then:

   (590) + If the Priority in the BACKUP ADVERTISEMENT is zero,
```

```
                then:

            (595) * Cancel Backup_Down_Timer.

            (600) * Remove the Peer from Peer table.

         (605) + else // priority non-zero

            (610) * Update the peer table with peer information.

            (615) * Set Backup_Adver_Interval to Adver Interval
            contained in the BACKUP ADVERTISEMENT.

            (620) * Recompute the Backup_Down_Interval.

            (625) * Reset the Backup_Down_Timer to Backup_Down_Interval.

         (630) +endif // was priority zero?

         (635) + Recalculate Critical_Backup.

         (640) + If Critical_BFD_Session exists and this
         instance is not the Critical_Backup:

            (645) * Tear Down the Critical_BFD_Session.

         (650) + else If Critical_BFD_Session doesnot exists and this
         instance is the Critical_Backup:

            (655) * BootStrap Critical_BFD_Session with Master.

         (660) + endif // Critical_Backup change

      (665) -endif // was backup advertisement recv?

      (670) - If Backup_Down_Timer fires, then:

         (675) + Remove the Peer from Peer table.

         (680) + If Critical_BFD_Session does not exist:

            (685) @ Recalculate Critical_Backup.

            (690) @ If This instance is the Critical_Backup:

               (695) +@ BootStrap Critical_BFD_Session with Master.

            (700) @ endif // Critical_Backup change
```

(705) + endif // Critical_BFD_Session does not exist?

(710) -endif // Backup_Down_Timer fires?

(715) - If Backup_Adver_Timer fires, then:

(720) + Send a BACKUP ADVERTISEMENT.

(725) + Reset the Backup_Adver_Timer to Backup_Advertisement_Interval.

(730) -endif // Backup_Down_Timer fires?

(735) endwhile // Backup state

3.6.3.3. Master

Following state machine replaces the state machine outlined in section 6.4.3 of [RFC5798]

(800) While in this state, a VRRP router MUST do the following:

(805) - If the protected IPvX address is an IPv4 address, then:

(810) + MUST respond to ARP requests for the IPv4 address(es) associated with the virtual router.

(815) - else // ipv6

(820) + MUST be a member of the Solicited-Node multicast address for the IPv6 address(es) associated with the virtual router.

(825) + MUST respond to ND Neighbor Solicitation message for the IPv6 address(es) associated with the virtual router.

(830) + MUST send ND Router Advertisements for the virtual router.

(835) + If Accept_Mode is False: MUST NOT drop IPv6 Neighbor Solicitations and Neighbor Advertisements.

(840) -endif // ipv4?

(845) - MUST forward packets with a destination link-layer MAC address equal to the virtual router MAC address.

(850) - MUST accept packets addressed to the IPvX address(es) associated with the virtual router if it is the IPvX address owner or if Accept_Mode is True. Otherwise, MUST NOT accept these packets.

(855) - If a Shutdown event is received, then:

(860) + Cancel the Adver_Timer.

(865) + Send an ADVERTISEMENT with Priority = 0,

(870) + Cancel Backup_Down_Timers.

(875) + Remove Peer table.

(880) + If Critical_BFD_Session Exists:

(885) * Tear down Critical_BFD_Session

(890) + endif // If Critical_BFD_Session Exists

(895) + Transition to the {Initialize} state.

(900) -endif // shutdown recv

(905) - If the Adver_Timer fires, then:

(910) + Send an ADVERTISEMENT.

(915) + Reset the Adver_Timer to Advertisement_Interval.

(920) -endif // advertisement timer fired

(925) - If an ADVERTISEMENT is received, then:

(930) -+ If the Priority in the ADVERTISEMENT is zero, then:

(935) -* Send an ADVERTISEMENT.

(940) -* Reset the Adver_Timer to Advertisement_Interval.

(945) -+ else // priority was non-zero

(950) -* If the Priority in the ADVERTISEMENT is greater than the local Priority,

(955) -* or

```
            (960) -* If the Priority in the ADVERTISEMENT is equal to
            the local Priority and the primary IPvX Address of the
            sender is greater than the local primary IPvX Address, then:

               (965) -@ Cancel Adver_Timer

               (970) -@ Set Master_Adver_Interval to Adver Interval
               contained in the ADVERTISEMENT

               (975) -@ Recompute the Skew_Time

               (980) @ Recompute the Master_Down_Interval

               (985) @ Set Master_Down_Timer to Master_Down_Interval

               (990) If Critical_BFD_Session Exists:

                  (995) @+ Tear Critical_BFD_Session

               (960) @ endif //Critical_BFD_Session Exists?

               (965) @ Calculate Critical_Backup.

               (970) @ If this instance is Critical_Backup:

                  (975) @+ BootStrap Critical_BFD_Session with new
                           Master.

               (980) @ endif // am i Critical_Backup?

               (985) @ Transition to the {Backup} state

            (990) * else // new Master logic

               (995) @ Discard ADVERTISEMENT

            (1000) *endif // new Master detected

         (1005) +endif // was priority zero?

      (1010) -endif // advert recv

      (1015) - If a BACKUP ADVERTISEMENT is received, then:

         (1020) + If the Priority in the BACKUP ADVERTISEMENT is
                  zero, then:

            (1025) * Remove the Peer from peer table.
```

```
         (1030) + else: // priority non-zero

            (1035) * Update the Peer info in peer table.

            (1040) * Recompute the Backup_Down_Interval

            (1045) * Reset the Backup_Down_Timer to
                     Backup_Down_Interval

         (1050) + endif // priority in backup advert zero

         (1055) + Calculate the Critical_Backup

         (1060) + If Critical_BFD_Session doesnot exist:

            (1065) * BootStrap Critical_BFD_Session

         (1070) + else if Critical_BFD_Session exist and
                  Critical_Backup changes:

            (1075) + Tear Critical_BFD_Session with old Backup

            (1080) + BootStrap Critical_BFD_Session with Critical_Backup

         (1085) + endif //  Critical_BFD_Session check?

      (1090) - endif // backup advert recv

      (1095) - If Critical_BFD_Session transitions from UP to DOWN,
      then:
         (1100) + Cancel Backup_Down_Timer

         (1105) + Delete the Peer info from peer table

         (1200) + Calculate the Critical_Backup

         (1205) + BootStrap Critical_BFD_Session with Critical_Backup

      (1210) - endif // BFD session transition

   (1215) endwhile // in Master
```

4. Applicability of p2mp BFD

[I-D.draft-ietf-bfd-multipoint] describes extensions to the BFD protocol for its use in multipoint and multicast networks. With these extensions p2mp BFD can support sub-second failure detection of

the root by tail nodes. In fact, a redundancy group may be viewed as p2mp BFD session with its Master being the root and Backup routers being tail nodes. Once Master selection process is completed, the Master router starts transmitting BFD control packets with IPvX address associated with the VRID as source IPvX address. Backup router demultiplexes p2mp BFD tail sessions based on IPvX source address associated with the virtual router that it been configured with. Once Backup router accepts p2mp session from the new Master router, the Backup router MAY use My Discriminator from received p2mp BFD control packet to demultiplex p2mp BFD sessions. When a Backup router detects failure of the Master router it re-evaluates its role in the VRID. As result, the Backup router may become the Master router of the given VRID or continue as a Backup router. If the former is the case, then the new Master router MUST select My Discriminator and start transmitting p2mp BFD control packets using Master IPvX address as source IPvX address for p2mp BFD control packets. If the latter is the case, then the Backup router MUST wait for p2mp BFD control packet with source IPvX address set to IPvX address associated with the VRID.

4.1. VRRP State Machine with p2mp BFD

Following section outlines the interaction between VRRP protocol [RFC5798] state machine and p2mp BFD. Please refer to the section 6.4 of [RFC5798] for State description.

4.1.1. Initialize

Following state machine replaces the state machine outlined in section 6.4.1 of [RFC5798]

(100) If a Startup event is received, then:

(105) - If the Priority = 255 (i.e., the router owns the IPvX address associated with the virtual router), then:

(110) + Send an ADVERTISEMENT

(115) + If the protected IPvX address is an IPv4 address, then:

(120) * Broadcast a gratuitous ARP request containing the virtual router MAC address for each IP address associated with the virtual router.

(125) + else // IPv6

(130) * For each IPv6 address associated with the virtual router, send an unsolicited ND Neighbor Advertisement with the Router Flag (R) set, the Solicited Flag (S) unset, the Override flag (O) set, the target address set to the IPv6 address of the virtual router, and the target link-layer address set to the virtual router MAC address.

(135) +endif // was protected addr IPv4?

(140) + Set the Adver_Timer to Advertisement_Interval

(145) + Transition to the {Master} state

(150) - else // rtr does not own virt addr

(155) + Set Master_Adver_Interval to Advertisement_Interval

(160) + Set the Master_Down_Timer to Master_Down_Interval

(165) + BootStrap BFD MultipointTail Session

(170) + Transition to the {Backup} state

(175) -endif // priority was not 255

(180) endif // startup event was recv

4.1.2. Backup

Following state machine replaces the state machine outlined in section 6.4.2 of [RFC5798]

(300) While in this state, a VRRP router MUST do the following:

```
(305) - If the protected IPvX address is an IPv4 address, then:

   (310) + MUST NOT respond to ARP requests for the IPv4
   address(es) associated with the virtual router.

(315) - else // protected addr is IPv6

   (320) + MUST NOT respond to ND Neighbor Solicitation messages
   for the IPv6 address(es) associated with the virtual router.

   (325) + MUST NOT send ND Router Advertisement messages for the
   virtual router.

(330) -endif // was protected addr IPv4?

(335) - MUST discard packets with a destination link-layer MAC
address equal to the virtual router MAC address.

(340) - MUST NOT accept packets addressed to the IPvX address(es)
associated with the virtual router.

(345) - If a Shutdown event is received, then:

   (350) + Cancel the Master_Down_Timer

   (355) + Transition to the {Initialize} state

(360) -endif // shutdown recv

(365) - If the Master_Down_Timer fires or
        BFD MultipointTail session transitions from UP to DOWN,
        then:

   (370) + Send an ADVERTISEMENT

   (375) + If the protected IPvX address is an IPv4 address, then:

      (380) * Broadcast a gratuitous ARP request on that interface
      containing the virtual router MAC address for each IPv4
      address associated with the virtual router.

    (385) + else // ipv6

      (390) * Compute and join the Solicited-Node multicast
      address [RFC4291] for the IPv6 address(es) associated with
      the virtual router.

      (395) * For each IPv6 address associated with the virtual
```

router, send an unsolicited ND Neighbor Advertisement with the Router Flag (R) set, the Solicited Flag (S) unset, the Override flag (O) set, the target address set to the IPv6 address of the virtual router, and the target link-layer address set to the virtual router MAC address.

(400) +endif // was protected addr ipv4?

(405) + Set the Adver_Timer to Advertisement_Interval

(410) + Tear down BFD MultipointTail Session

(415) + BootStrap BFD MultipointHead Session

(420) + Transition to the {Master} state

(425) -endif // Master_Down_Timer fired

(430) - If an ADVERTISEMENT is received, then:

(435) + If the Priority in the ADVERTISEMENT is zero, then:

(440) * Set the Master_Down_Timer to Skew_Time

(445) + else // priority non-zero

(450) * If Preempt_Mode is False, or if the Priority in the ADVERTISEMENT is greater than or equal to the local Priority, then:

(455) @ Set Master_Adver_Interval to Adver Interval contained in the ADVERTISEMENT

(460) @ Recompute the Master_Down_Interval

(465) @ Reset the Master_Down_Timer to Master_Down_Interval

(470) * else // preempt was true or priority was less

(475) @ Discard the ADVERTISEMENT

(480) *endif // preempt test

(485) +endif // was priority zero?

(490) -endif // was advertisement recv?

(495) endwhile // Backup state

4.1.3. Master

Following state machine replaces the state machine outlined in section 6.4.3 of [RFC5798]

(600) While in this state, a VRRP router MUST do the following:

(605) - If the protected IPvX address is an IPv4 address, then:

(610) + MUST respond to ARP requests for the IPv4 address(es) associated with the virtual router.

(615) - else // ipv6

(620) + MUST be a member of the Solicited-Node multicast address for the IPv6 address(es) associated with the virtual router.

(625) + MUST respond to ND Neighbor Solicitation message for the IPv6 address(es) associated with the virtual router.

(630) ++ MUST send ND Router Advertisements for the virtual router.

(635) ++ If Accept_Mode is False: MUST NOT drop IPv6 Neighbor Solicitations and Neighbor Advertisements.

(640) +-endif // ipv4?

(645) - MUST forward packets with a destination link-layer MAC address equal to the virtual router MAC address.

(650) - MUST accept packets addressed to the IPvX address(es) associated with the virtual router if it is the IPvX address owner or if Accept_Mode is True. Otherwise, MUST NOT accept these packets.

(655) - If a Shutdown event is received, then:

(660) + Cancel the Adver_Timer

(665) + Send an ADVERTISEMENT with Priority = 0

(670) + Tear down BFD MultipointHead Session

(675) + Transition to the {Initialize} state

(680) -endif // shutdown recv

(685) - If the Adver_Timer fires, then:

(690) + Send an ADVERTISEMENT

(695) + Reset the Adver_Timer to Advertisement_Interval

(700) -endif // advertisement timer fired

(705) - If an ADVERTISEMENT is received, then:

(710) -+ If the Priority in the ADVERTISEMENT is zero, then:

(715) -* Send an ADVERTISEMENT

(720) -* Reset the Adver_Timer to Advertisement_Interval

(725) -+ else // priority was non-zero

(730) -* If the Priority in the ADVERTISEMENT is greater than the local Priority,

(735) -* or

(740) -* If the Priority in the ADVERTISEMENT is equal to the local Priority and the primary IPvX Address of the sender is greater than the local primary IPvX Address, then:

(745) -@ Cancel Adver_Timer

(750) -@ Set Master_Adver_Interval to Adver Interval contained in the ADVERTISEMENT

(755) -@ Recompute the Skew_Time

(760) @ Recompute the Master_Down_Interval

(765) @ Set Master_Down_Timer to Master_Down_Interval

(770) + Tear down BFD MultipointHead Session

(775) + BootStrap BFD MultipointTail Session

(780) @ Transition to the {Backup} state

(785) * else // new Master logic

```
                (790) @ Discard ADVERTISEMENT

             (795) *endif // new Master detected

          (800) +endif // was priority zero?

       (805) -endif // advert recv

   (810) endwhile // in Master
```

5. Scalability Considerations

To reduce the number of packets generated at a regular interval, Backup Advert packets may be sent at a reduced rate as compared to Advert packets sent by the VRRP Master.

In a Data Centre with VXLAN extending the Layer 2 network, when implementing Section 4 of this document, inherently multicast traffic is flooded or replicated to all the Virtual Tunneling End Points by means of multicast traffic in the underlay network. The amount of replication or flooding depends on the number of Virtual Tunneling End Points connected to the VXLAN network. VRRP is typically deployed on the Virtual Tunneling End Points. If Multipoint BFD is used for tracking the state of VRRP Master Router the Multipoint BFD packets will get carried over the Layer 2 Overlay, this can lead to a lot of traffic getting flooded on the overlay as the rate at which BFD packets are generated will be typically in sub second range. Which is the problem if VRRP is configured with sub second timers. So in such scenarios where flooding of Multicast traffic is a concern, it is recommended to use Point to Point BFD sessions to avoid inherent flooding of Multicast traffic and configure VRRP to default or slow timers.

6. Operational Considerations

A VRRP peer that forms a member of this Virtual Router, but does not support this feature or extension must be configured with the lowest priority, and will only operate as the Router of last resort on failure of all other VRRP routers supporting this functionality.

It is recommended that mechanism defined by this draft, to interface VRRP with BFD should be used when BFD can support more aggressive monitoring timers than VRRP. Otherwise it is desirable not to interface VRRP with BFD for determining the health of VRRP Master.

This Draft does not preclude the possibility of the peer table being populated by means of manual configuration, instead of using the BACKUP ADVERTISEMENT as defined by the Draft.

7. IANA Considerations

This document requests IANA to create a new name space that is to be managed by IANA. The document defines a new VRRP Packet Type. The VRRP Packet Types are discussed below.

a) Type 1 (ADVERTISEMENT) defined in section 5.2.2 of [RFC5798]
b) Type 2 (BACKUP ADVERTISEMENT) defined in section 3.3 of this document

7.1. A New Name Space for VRRP Packet Types

This document defines in Section 3.3 a "BACKUP ADVERTISEMENT" VRRP Packet Type. The new name space has to be created by the IANA and they will maintain this new name space. The field for this namespace is 4-Bits, and IANA guidelines for assignments for this field are as follows:

ADVERTISEMENT	1
BACKUP ADVERTISEMENT	2

Future allocations of values in this name space are to be assigned by IANA using the "Specification Required" policy defined in [IANA-CONS]

8. Security Considerations

Security considerations discussed in [RFC5798], [RFC5880] and [I-D.draft-ietf-bfd-multipoint], apply to this document. There are no additional security considerations identified by this draft.

9. Acknowledgements

The authors gratefully acknowledge the contributions of Gerry Meyer, and Mouli Chandramouli, for their contributions to the draft. The authors will also like to thank Jeffrey Haas, Maik Pfeil, Chris Bowers and Vengada Prasad Govindan for their comments and suggestions.

10. Normative References

[RFC5880] Katz, D. and D. Ward, "Bidirectional Forwarding Detection (BFD)", RFC 5880, 2010.

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

[RFC5881] Katz, D. and D. Ward, "Bidirectional Forwarding Detection (BFD) for IPv4 and IPv6 (Single Hop)", RFC 5881, 2010.

[RFC5798] Nadas, S., "Virtual Router Redundancy Protocol (VRRP) Version 3 for IPv4 and IPv6", RFC 5798, 2010.

[I-D.draft-ietf-bfd-multipoint] Katz, D., Ward, D., and S. Pallagatti, "BFD for Multipoint Networks", Work in Progress draft-ietf-bfd-multipoint-07, 2015.

[IANA-CONS] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", RFC 2434, 1998.

Authors' Addresses

Nitish Gupta
Cisco Systems, Inc.
Sarjapur Outer Ring Road
Bangalore 560103
India

Phone: +91 80 4429 2530
Email: nitisgup@cisco.com
URI: http://www.cisco.com/

Aditya Dogra
Cisco Systems, Inc.
Sarjapur Outer Ring Road
Bangalore 560103
India

Phone: +91 80 4429 2166
Email: addogra@cisco.com
URI: http://www.cisco.com/

Colin Docherty
25 George Grieve Way
Tranent
East Lothian, Scotland EH332QT
United Kingdom

Email: colin@doch.org.uk

Greg Mirsky
Ericsson

Email: gregory.mirsky@ericsson.com

Jeff Tantsura
Individual

Email: jefftant.ietf@gmail.com

Network Working Group
Internet-Draft
Intended status: Informational
Expires: January 6, 2016

A. Shaikh
J. George
Google
R. Shakir
BT
F. Chen
Verizon Communications

July 5, 2015

A Data Model for Locally Generated Routes in Service Provider Networks
draft-openconfig-rtgwg-local-routing-00

Abstract

This document defines a YANG data model for configuring and managing locally generated routes in a vendor-neutral way and based on operational best practice. Locally generated routes are those created by configuration, rather than by dynamic routing protocols. Such routes include static routes, locally created aggregate routes for reducing the number of constituent routes that must be advertised, summary routes for IGPs, etc.

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 January 6, 2016.

1. Introduction

This document describes a simple YANG [RFC6020] data model for locally generated routes, i.e., those not created by dynamic routing protocols. These include static routes, locally created aggregate routes, summary routes for IGPs, etc.

1.1. Goals and approach

This model expresses locally generated routes as generically as possible, avoiding configuration of protocol-specific attributes at the time of route creation. This is primarily to avoid assumptions about how underlying router implementations handle route attributes in various routing table data structures they maintain. Hence, the definition of locally generated routes essentially creates 'bare' routes that do not have any protocol- specific attributes.

When protocol-specific attributes must be attached to a route (e.g., communities on a locally defined route meant to be advertised via BGP), the attributes should be attached via a protocol-specific policy after importing the route into the protocol for distribution (again via routing policy). This model is intended to be used with the generic routing policy framework defined in [I-D.shaikh-rtgwg-policy-model].

This model does not aim to be feature complete -- it is a subset of the configuration parameters available in a variety of vendor implementations, but supports widely used constructs for managing local routes. Model development has been primarily driven by examination of actual configurations across operator networks.

While this document provides an overview of the model, the most current and authoritative version is available in the public YANG model repository [YANG-REPO].

2. Model overview

In this initial version of the local routing model, two primary types of locally generated routes are supported, static routes and local aggregates. Static routes are manually configured routes with a defined prefix and next-hop. The model support next-hops specified

as an IP address, interface, or a special defined value, e.g., DISCARD to prevent forwarding for the specified prefix. Aggregate routes are used to summarize constituent routes and avoid having to advertise each constituent individually, and hence increase routing table size. Aggregate routes are not used for forwarding, rather they are "activated" when a constituent route with a valid next hop is present in the IP routing table.

The model structure is shown below:

```
    +--rw local-routes
      +--rw config
      +--ro state
      +--rw static-routes
      |  +--rw static* [prefix]
      |        ...
      +--rw local-aggregates
         +--rw aggregate* [prefix]
               ...
```

Note that the model follows the convention of representing configuration and operational state at the leaves of the tree, based on recommendations in [I-D.openconfig-netmod-opstate]. In this case, the operational state consists primarily of the applied configuration.

3. Interaction with routing policy

In many vendor implementations, local routes may be annotated with various route attributes or policies. For example, when creating a locally generated aggregate route, it is often possible to specify BGP [RFC4271] communities which can then be used when filtering the routes sent to particular neighbors or peer groups. IGP implementations such as IS-IS and OSPF have similar capability to create "summary" routes that serve a similar purpose to BGP aggregates.

Since these and other local routes are conceptually similar from an operator standpoint, there is a desire to create a single model that may be used generically to address a number of use cases. The approach taken in this model is to define locally generated routes as "bare" routes, i.e., without any protocol-specific attributes such as BGP communities, IGP tags, etc. Instead, these attributes are expected to be added via policy when locally generated routes are injected into a particular protocol for subsequent advertisement.

Another important motivation for not including protocol-specific attributes when configuring local routes is that it assumes

implementations can support the association of a variety of attributes with a route. While this may be true in some implementations, others may segregate routing tables for different protocols into different data structures such that it would not be possible to attach attributes from, say BGP, onto an OSPF route. For this reason, we constrain attributes on locally generated routes to be attached via policy as they are imported into different protocols.

For example, consider the case of configuring a new prefix to be advertised to neighbors via BGP. Using the local routing model and the routing policy model in [I-D.shaikh-rtgwg-policy-model], one way to do this is enumerated below:

1. Declare a static route for the advertised prefix using the local routing model (e.g., with a next hop set to DISCARD). This route is placed in the IP routing table.

2. Define a policy to add BGP attributes to the route -- the policy would match on the prefix and the origin of the route (i.e., STATIC) and its action could be to add a BGP community.

3. Apply the BGP policy as an import policy within BGP, e.g., using the apply-policy statement in the policy model, to import the route into BGP.

4. Export the route to neighbors using an export policy as usual, filtering on the added community attribute if appropriate.

The step of creating a policy to add BGP (or other protocol-specific) attributes to the route is optional if an operator simply wishes to export the route to neighbors, as opposed to also filtering on attributes that are assumed to be present on the locally generated route.

This version of the model does support the capability to specify a generic route tag that can be associated with locally generated routes (i.e., both statics and aggregates). The tag can be used to filter routes that are imported into different routing protocols, for example, or to control which routes are exported to a neighbor. The route tagging capability may be refined further as more implementor feedback is incorporated into the model.

4. Security Considerations

Routing configuration has a significant impact on network operations, and as such any related model carries potential security risks.

YANG data models are generally designed to be used with the NETCONF protocol over an SSH transport. This provides an authenticated and secure channel over which to transfer configuration and operational data. Note that use of alternate transport or data encoding (e.g., JSON over HTTPS) would require similar mechanisms for authenticating and securing access to configuration data.

Most of the data elements in the local routing model could be considered sensitive from a security standpoint. Unauthorized access or invalid data could cause major disruption.

5. IANA Considerations

This YANG data model and the component modules currently use a temporary ad-hoc namespace. If and when it is placed on redirected for the standards track, an appropriate namespace URI will be registered in the IETF XML Registry" [RFC3688]. The routing policy YANG modules will be registered in the "YANG Module Names" registry [RFC6020].

6. YANG module

The local routing model is described by the YANG module below.

```
<CODE BEGINS> file local-routing.yang
module local-routing {

  yang-version "1";

  // namespace
  namespace "http://openconfig.net/yang/local-routing";

  prefix "loc-rt";

  // import some basic types
  import ietf-inet-types { prefix inet; }
  import policy-types { prefix pt; }


  // meta
  organization "OpenConfig working group";

  contact
    "OpenConfig working group
    www.openconfig.net";

  description
    "This module describes configuration and operational state data
```

```
      for routes that are locally generated, i.e., not created by
      dynamic routing protocols.  These include static routes, locally
      created aggregate routes for reducing the number of constituent
      routes that must be advertised, summary routes for IGPs, etc.

      This model expresses locally generated routes as generically as
      possible, avoiding configuration of protocol-specific attributes
      at the time of route creation.  This is primarily to avoid
      assumptions about how underlying router implementations handle
      route attributes in various routing table data structures they
      maintain.  Hence, the definition of locally generated routes
      essentially creates 'bare' routes that do not have any protocol-
      specific attributes.

      When protocol-specific attributes must be attached to a route
      (e.g., communities on a locally defined route meant to be
      advertised via BGP), the attributes should be attached via a
      protocol-specific policy after importing the route into the
      protocol for distribution (again via routing policy).";

    revision "2015-05-01" {
      description
        "Initial revision";
      reference "TBD";
    }

    // extension statements

    // feature statements

    // identity statements

    // typedef statements

    typedef local-defined-next-hop {
      type enumeration {
        enum DROP {
          description
            "Discard or black-hole traffic for the corresponding
            destination";
        }
      }
      description
        "Pre-defined next-hop designation for locally generated
        routes";
    }

    // grouping statements
```

```
    grouping local-generic-settings {
      description
        "Generic options that can be set on local routes When
        they are defined";

      leaf set-tag {
        type pt:tag-type;
        description
          "Set a generic tag value on the route. This tag can be
          used for filtering routes that are distributed to other
          routing protocols.";
      }
    }

    grouping local-static-config {
      description
        "Configuration data for static routes.";

      leaf prefix {
        type inet:ip-prefix;
        description
          "Destination prefix for the static route, either IPv4 or
          IPv6.";
      }

      leaf-list next-hop {
        type union {
          type inet:ip-address;
          type local-defined-next-hop;
          type string;
          //TODO: this should be a leafref pointing to a configured
          //interface, but YANG 1.0 does not support leafrefs in a
          //union type.  It should be updated when YANG 1.1 is
          //released.
        }
        description
          "Specify a set of next hops.  Each entry may be an IP
          address, interface, or a single pre-defined next-hop can be
          used, e.g., drop";
      }

      uses local-generic-settings;
    }

    grouping local-static-state {
      description
        "Operational state data for static routes";
    }
```

```
     grouping local-static-top {
       description
         "Top-level grouping for the list of static route definitions";

       container static-routes {
         description
           "Enclosing container for the list of static routes";

         list static {
           key prefix;
           description
             "List of locally configured static routes";

           leaf prefix {
             type leafref {
               path "../config/prefix";
             }
             description
               "Reference to the destination prefix for the static
               route";
           }

           container config {
             description
               "Configuration data for static routes";

             uses local-static-config;
           }

           container state {

             config false;

             description
               "Operational state data for static routes";

             uses local-static-config;
             uses local-static-state;
           }
         }
       }
     }

     grouping local-aggregate-config {
       description
         "Configuration data for aggregate routes";

       leaf prefix {
```

```
        type inet:ip-prefix;
        description
          "Aggregate prefix to be advertised";
      }

      leaf discard {
        type boolean;
        default false;
        description
          "When true, install the aggregate route with a discard
          next-hop -- traffic destined to the aggregate will be
          discarded with no ICMP message generated.  When false,
          traffic destined to an aggregate address when no
          constituent routes are present will generate an ICMP
          unreachable message.";
      }

      uses local-generic-settings;

    }

    grouping local-aggregate-state {
      description
        "Operational state data for local aggregate advertisement
        definitions";
    }

    grouping local-aggregate-top {
      description
        "Top-level grouping for local aggregates";

      container local-aggregates {
        description
          "Enclosing container for locally-defined aggregate
          routes";

        list aggregate {
          key prefix;
          description
            "List of aggregates";

          leaf prefix {
            type leafref {
              path "../config/prefix";
            }
            description
              "Reference to the configured prefix for this aggregate";
          }
```

```
          container config {
            description
              "Configuration data for aggregate advertisements";

            uses local-aggregate-config;
          }

          container state {

            config false;

            description
              "Operational state data for aggregate
              advertisements";

            uses local-aggregate-config;
            uses local-aggregate-state;
          }
        }
      }
    }

    grouping local-routes-config {
      description
        "Configuration data for locally defined routes";
    }

    grouping local-routes-state {
      description
        "Operational state data for locally defined routes";
    }

    grouping local-routes-top {
      description
        "Top-level grouping for local routes";

      container local-routes {
        description
          "Top-level container for local routes";

        container config {
          description
            "Configuration data for locally defined routes";

          uses local-routes-config;
        }

        container state {
```

```
          config false;

          description
            "Operational state data for locally defined routes";

          uses local-routes-config;
          uses local-routes-state;
        }

        uses local-static-top;
        uses local-aggregate-top;
      }
    }

    uses local-routes-top;

  }
  <CODE ENDS>
```

7. Examples

Below we show an example of XML-encoded configuration data using the local routing model, in conjunction with the policy [I-D.shaikh-rtgwg-policy-model] and BGP [I-D.shaikh-idr-bgp-model] models to illustrate how local aggregate routes are defined and distributed into protocols. Although the example focuses on BGP, the aggregate routes may be used with other routing protocols (e.g., IGPs) as mentioned in Section Section 3. Note that the XML has been modified to improve readability.

```
<!-- define an aggregate route -->
<local-routing>
  <local-routes>
    <local-aggregates>
      <aggregate>
        <prefix>10.185.224.0/19</prefix>
        <config>
          <prefix>10.185.224.0/19</prefix>
          <discard>true</discard>
        </config>
      </aggregate>
    </local-aggregates>
  </local-routes>
</local-routing>

<routing-policy>
```

```
    <defined-sets>
      <prefix-sets>
        <prefix-set>
          <prefix-set-name>AGGR_INTERNAL</prefix-set-name>
            <prefix>
              <ip-prefix>10.185.224.0/19</ip-prefix>
              <masklength-range>exact</masklength-range>
            </prefix>
        </prefix-set>
      </prefix-sets>

      <bgp-defined-sets>
        <community-sets>
          <community-set>
            <community-set-name>AGGR_BGP_COMM</community-set-name>
            <community-member>65532:10100</community-member>
            <community-member>65534:60110</community-member>
          </community-set>
        </community-sets>
      </bgp-defined-sets>
    </defined-sets>

    <policy-definitions>

      <!--policy definition to add BGP attributes to the route -->
      <policy-definition>
        <name>ADD_ATTR_BGP_AGGR</name>
        <statements>
          <statement>
            <name>statement-1</name>
            <conditions>
              <install-protocol-eq>LOCAL-AGGREGATE</install-protocol-eq>
              <match-prefix-set>
                  <prefix-set>AGGR_INTERNAL</prefix-set>
              </match-prefix-set>
            </conditions>
            <actions>
              <bgp-actions>
                <set-community>
                  <community-set-ref>AGGR_BGP_COMM</community-set-ref>
                </set-community>
              </bgp-actions>
              <accept-route />
            </actions>
          </statement>
        </statements>
      </policy-definition>
```

```
      <!-- create a policy to export the
      aggregate to UPSTREAM neighbors -->
      <policy-definition>
        <name>EDGE_EXPORT</name>
        <statements>
          <statement>
            <name>statement-2</name>
            <conditions>
              <install-protocol-eq>LOCAL-AGGREGATE</install-protocol-eq>
              <match-prefix-set>
                  <prefix-set>AGGR_INTERNAL</prefix-set>
              </match-prefix-set>
            </conditions>
            <actions>
              <bgp-actions>
                <set-med>100</set-med>
              </bgp-actions>
              <accept-route />
            </actions>
          </statement>
        </statements>
      </policy-definition>
    </policy-definitions>

  </routing-policy>


  <bgp>
    <global>
      <config>
        <as>65518</as>
      </config>
      <!-- apply a policy to import the aggregate into BGP -->
      <apply-policy>
        <config>
          <import-policy>ADD_ATTR_BGP_AGGR</import-policy>
        </config>
      </apply-policy>
    </global>
    <peer-groups>
      <peer-group>
        <peer-group-name>UPSTREAM</peer-group-name>
        <!-- apply a policy to advertise the
        aggregate to the UPSTREAM group -->
        <apply-policy>
          <config>
            <export-policy>EDGE_EXPORT</export-policy>
            <default-export-policy>REJECT_ROUTE</default-export-policy>
```

```
          </config>
        </apply-policy>
      </peer-group>
    </peer-groups>
  </bgp>
```

8. References

8.1. Normative references

[RFC6020] Bjorklund, M., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, October 2014.

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

[RFC3688] Mealling, M., "The IETF XML Registry", RFC 3688, January 2004.

8.2. Informative references

[YANG-REPO] "A Data Model for Locally Generated Routes in Service Provider Networks", July 2015, <https://github.com/YangModels/yang/tree/master/experimental/openconfig/local-routing>.

[I-D.shaikh-rtgwg-policy-model] Shaikh, A., Shakir, R., D'Souza, K., and C. Chase, "Routing Policy Configuration Model for Service Provider Networks", draft-shaikh-rtgwg-policy-model-01 (work in progress), July 2015.

[I-D.openconfig-netmod-opstate] Shakir, R., Shaikh, A., and M. Hines, "Consistent Modeling of Operational State Data in YANG", draft-openconfig-netmod-opstate-00 (work in progress), March 2015.

[I-D.shaikh-idr-bgp-model] Shaikh, A., Shakir, R., Patel, K., Hares, S., D'Souza, K., Bansal, D., Clemm, A., Alex, A., Jethanandani, M., and X. Liu, "BGP Model for Service Provider Networks", draft-shaikh-idr-bgp-model-02 (work in progress), June 2015.

Appendix A. Acknowledgements

The authors are grateful for valuable contributions to this document and the associated models from: Phil Bedard, Kevin Brintnall, Matt John, Marcus Hines, and Eric Osborne.

Authors' Addresses

Anees Shaikh
Google
1600 Amphitheatre Pkwy
Mountain View, CA 94043
US

Email: aashaikh@google.com

Joshua George
Google
1600 Amphitheatre Pkwy
Mountain View, CA 94043
US

Email: jgeorge@google.com

Rob Shakir
BT
pp. C3L, BT Centre
81, Newgate Street
London EC1A 7AJ
UK

Email: rob.shakir@bt.com
URI: http://www.bt.com/

Feihong Chen
Verizon Communications
40 Sylvan Rd.
Waltham, MA
US

Email: feihong.x.chen@verizon.com

Network Working Group
Internet-Draft
Intended status: Informational
Expires: May 7, 2016

R. Shakir
Jive Communications

November 4, 2015

A Data Model for Network Instances in Service Provider Networks
draft-openconfig-rtgwg-network-instance-01

Abstract

This document defines a YANG data model for configuring and managing forwarding instances on a network device - focused primarily on deployments in service provider networks. The model is vendor-neutral and based on operational requirements expressed by operators. A 'network instance' is defined to be a discrete forwarding table on a network element, which contains Layer 3 and/or Layer 2 forwarding entries. Each network instance may instantiate its own sets of protocols which control installation of forwarding entries into one or more tables (which may be Layer 2 FIBs, or Layer 3 RIBs).

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 May 7, 2016.

1. Introduction

Within service provider networks, it is typical for a network element to maintain multiple forwarding tables, allowing the network to support multiple topologies. For instance, a service provider may offer Layer 2 or Layer 3 VPN services to its customers - which requires a forwarding table per customer to be maintained on a network element. In other cases, multiple L3 services may be deployed to allow services such as Internet connectivity and private topologies to be maintained on a device.

This YANG [RFC6020] data model defines a construct which can be used to configure and retrieve operational state related to such instances.

1.1. Goals and Approach

The concept of maintaining multiple Layer 2 virtual switch instances (VSIs) and/or multiple Layer 3 virtual routing and forwarding (VRF) tables is common on network elements that are deployed in service provider networks. However, within such networks there are requirements for a network instance to run both Layer 2 and Layer 3 routing protocols, or contain routed interfaces (e.g., a VPLS service where there is a Layer 3 interface within it, for subscriber termination or to act as a default gateway). For this reason, the intention of the model presented here is to define a generic construct which allows isolation of a forwarding table which may contain Layer 3 RIB entries, Layer 2 FIB entries, or a combination both. This differs to the approach taken in other models, which tend to focus on solely L2 VSIs or L3 VRFs. An instance within the model is referred to as network instance - and may be configured to support L3 RIB entries only (i.e., be functionally equivalent to a VRF), L2 FIB entries only (i.e., act as a VSI) or support a combination of both.

The model presented in this document is explicitly biased towards deployments on service provider network equipment, as discussed between members of the OpenConfig project.

2. Model overview

```
module: openconfig-network-instance
   +--rw network-instances
      +--rw network-instance* [name]
         +--rw name                          -> ../config/name
         +--rw config
            ...
         +--ro state
            ...
         +--rw inter-instance-policies
         |  +--rw apply-policy
            ...
         +--rw table-connections
         |  +--rw table-connection* [src-table dst-table]
         |     +--rw src-table       -> ../config/src-table
         |     +--rw dst-table       -> ../config/dst-table
                ...
         |     +--rw apply-policy
                ...
         +--rw tables
         |  +--rw table* [table-name]
                ...
         +--rw interfaces
                ....
         +--rw connection-points
         |  +--rw connection-point* [connection-point-id]
              ...
         +--rw protocols
            +--rw protocol* [identifier name]
                ...
               +--rw static
                ...
               +--rw aggregate
                ...
```

The key elements of the network instance model shown in the above outline tree view are:

- Type of network instance and key configuration parameters that relate to it (e.g., configuration of a route distinguisher for L3 routing instance).

- Policies which define how the instance exchanges routes with other instances - for example, allowing forwarding entries to be leaked between a global and private network instance.

- The individual tables that are maintained by a network instance. This caters for the approach whereby an implementation maintains per-protocol tables - and it is possible to examine these tables separately. The choice to implement this table definition is an open issue - as discussed in Section 3.1. Along with this table definition, a set of policies determining how entries are leaked between the tables is defined.

- Connections between the different tables that are maintained by the network instance. Where protocols populate individual tables, such connections facilitate leaking of routes between individual protocols (by means of populating entries from one into the table of the other).

- The interfaces associated with a certain network instance - the current implementation assumption (as demonstrated in the openconfig-interfaces models) is that configuration parameters that relate directly to the interface (e.g., IP addressing) are configured directly under the interface construct. This requires that a network instance must have a means to be able to associate itself with an interface.

- A logical grouping of local or remote interfaces into 'connection points' utilised by a number of Layer 2 forwarding approaches (e.g., redundancy for attachment points for PWE or L2VPN services). This allows interfaces which are associated with the network instance to be associated with one another, and hence referenced as a group. This construct can be used to represent redundant interfaces for a P2P L2VPN - and is extensible to support Ethernet segments for other L2VPN types.

- A list of the protocols that are instantiated under the network instance. The protocols that are defined in [I-D.openconfig-rtgwg-local-routing] are included directly such that forwarding entries generated as static or aggregate routes can be matched in a routing policy. Other protocols, such as BGP (for which the data model is defined in [I-D.ietf-idr-bgp-model] would augment this list to include their own configuration options). It should be noted that the list of protocols is not keyed directly on the protocol name, but rather also includes an identifier for that instance. This supports cases where a single network instance may run multiple instances of a protocol.

It is expected that all devices maintain at least one network instance, which represents the default (or global) forwarding table of the device. Additional instances are intended to support virtual instances, such as VSIs and VRFs.

Where base configuration is required for such instances to be utilised (e.g., an route distinguisher value is required) then this is configured globally for each network instance. It expected that additional parameters, such as the encapsulation utilised for the instance will be added in future revisions of the model.

Policies for leaking of routes between instances and tables leverage [I-D.ietf-rtgwg-policy-model] as a policy framework. This allows definition of 'import' policies (allowing a pull model) or export policies (allowing a push model) to be utilised for installation of entries into a local or remote network instance. The same framework is utilised to leak prefixes between multiple tables that are instantiated under an individual network instance.

2.1. Supporting Layer 2 Forwarding Constructs

Whilst the initial focus of parameters in the model is to allow the instantiation of attributes related to Layer 3 forwarding constructs - it is expected that other configuration parameters, such as the definition of 'cross-connections' between local and/or remote interfaces or connection points can be utilised to represent forwarding constructs within the network instance.

The addition of this to the model is still a work-in-progress. The inclusion of the connection point configuration and state parameters is intended to demonstrate how L2 and L3 constructs can co-exist with one another within the model.

3. Open Issues

3.1. Representation of per-protocol tables vs. a single table for all protocols

Some implementations maintain a single table into which all protocols install entries. Each protocol is then configured to explicitly import entries from that table into their local 'export' table. Others maintain multiple RIBs and require explicit connections to be made between these tables such that entries from one protocol are made available to another.

There are essentially two alternatives for how this is modelled in the network-instance model:

1. Allow each protocol to specify a 'target-table' that indicates the table that its entries are to be installed into. In the case that the same target table is specified for multiple protocols, implicit import/export configuration between tables could be created on implementations that implement multiple RIBs.

Explicit configuration can be utilised to explicitly allow an operator to leak between tables in the case that multiple target tables are specified. It should be noted that in fact, protocols that support multiple address families need to allow specification of the target table on a per-address-family basis.

2. Assume that all protocols implement their own target tables. In this case, explicit configuration is utilised to leak between them. Implementations that do not support such per-protocol tables could implicitly generate the configuration that results in forwarding entries being leaked between the tables such that the appearance of a single target-table is maintained.

As per the discussion in Section 2 currently, this model chooses the former implementation approach.

4. Security Considerations

Routing configuration has a significant impact on network operations, and as such any related model carries potential security risks.

YANG data models are generally designed to be used with the NETCONF protocol over an SSH transport. This provides an authenticated and secure channel over which to transfer configuration and operational data. Note that use of alternate transport or data encoding (e.g., JSON over HTTPS) would require similar mechanisms for authenticating and securing access to configuration data.

Most of the data elements in the network intsance model could be considered sensitive from a security standpoint. Unauthorized access or invalid data could cause major disruption.

5. IANA Considerations

This YANG data model and the component modules currently use a temporary ad-hoc namespace. If and when it is placed on redirected for the standards track, an appropriate namespace URI will be registered in the IETF XML Registry" [RFC3688]. The routing policy YANG modules will be registered in the "YANG Module Names" registry [RFC6020].

6. YANG module

The network instance model is described by the YANG module below.

<CODE BEGINS> file openconfig-network-instance.yang

```
module openconfig-network-instance {

  yang-version "1";

  // namespace
  namespace "http://openconfig.net/yang/network-instance";

  prefix "netinst";

  // import some basic types
  //import ietf-inet-types { prefix inet; }
  import ietf-yang-types { prefix "yang"; }
  import ietf-inet-types { prefix "inet"; }
  import openconfig-network-instance-types { prefix "nit"; }
  import openconfig-policy-types { prefix "pt"; }
  import openconfig-routing-policy { prefix "rpol"; }
  import openconfig-local-routing { prefix "lroute"; }
  import openconfig-interfaces { prefix "ocif"; }
  import openconfig-extensions { prefix "ocext"; }

  // meta
  organization "OpenConfig working group";

  contact
    "OpenConfig working group
    www.openconfig.net";

  description
    "An OpenConfig description of a network-instance. This may be
    a Layer 3 forwarding construct such as a virtual routing and
    forwarding (VRF) instance, or a Layer 2 instance such as a
    virtual switch instance (VSI). Mixed Layer 2 and Layer 3
    instances are also supported.";

  ocext:openconfig-version "0.1.0";

  revision "2015-10-18" {
    description
      "Initial revision";
    reference "0.1.0";
  }

  grouping network-instance-top {
    description
      "Top-level grouping containing a list of network instances.";

        container network-instances {
      description
```

```
        "The L2, L3, or L2+L3 forwarding instances that are
        configured on the local system";

                list network-instance {
                        key "name";

        description
          "Network instances configured on the local system";

                        leaf name {
                                type leafref {
                                        path "../config/name";
                                }
                                description
                                        "A unique name identifying the network i
nstance";
                        }

        container config {
          description
            "Configuration parameters relating to a network
            instance";
          uses network-instance-config;
          uses network-instance-l3vrf-config {
            when "../type = 'L3VRF'" {
              description
                "Layer 3 VRF configuration parameters included when a
                network instance is of type L3VRF";
            }
          }
        }
        container state {
          config false;
          description
            "Operational state parameters relating to a network
            instance";
          uses network-instance-config;
          uses network-instance-l3vrf-config {
            when "../type = 'L3VRF'" {
              description
                "Layer 3 VRF configuration parameters included when a
                network instance is of type L3VRF";
            }
          }
          uses network-instance-state;
        }

        container inter-instance-policies {
          description
```

```
            "Policies dictating how RIB or FIB entries are imported
            to and exported from this instance";

          uses rpol:apply-policy-group;
        }

        container table-connections {
          description
            "Policies dictating how RIB or FIB entries are propagated
            between tables";

          list table-connection {
            key "src-table dst-table";

            description
              "A list of connections between pairs of routing or
              forwarding tables, the leaking of entries between
              which is specified by the import and export policy";

            leaf src-table {
              type leafref {
                path "../config/src-table";
              }
              description
                "The name of the table which should be utilised
                as the source of forwarding or routing information";
            }

            leaf dst-table {
              type leafref {
                path "../config/dst-table";
              }
              description
                "The table to which routing entries should be
                exported";
            }

            container config {
              description
                "Configuration parameters relating to the connection
                between tables";
              uses inter-table-policies-config;
            }
            container state {
              config false;
              description
                "State parameters relating to the connection between
                tables";
```

```
            uses inter-table-policies-config;
          }

          uses rpol:apply-policy-group;
        }
      }

      container tables {
        description
          "The routing tables that are managed by this network
          instance";

        list table {
          key "table-name";

          description
            "A network instance manages one or more forwarding or
            routing tables. These may reflect the Layer 2
            forwarding information base, the Layer 3 routing
            information base of the MPLS LFIB. Protocols may be
            explictly associated with a particular table into
            which they populate entries. Multiple protocols may
            install entries into a single table, or there may be a
            1:1 relationship between a routing protocol and a
            table .The import-policy and export-policy lists are
            used to specify how routes leak between different
            tables within the same forwarding instance.";

          leaf table-name {
            type leafref {
              path "../config/table-name";
            }
            description
              "A name for the table";
          }

          container config {
            description
              "Configuration parameters related to the table";
            uses table-config;
          }

          container state {
            config false;
            description
              "State parameters related to the table";
            uses table-config;
          }
```

```
    }
  }

  container interfaces {
    description
      "Interfaces associated with this network intance";

    container config {
      description
        "Configuration parameters relating to interfaces
        associated with the instance";
      uses instance-interfaces-config;
    }
    container state {
      config false;
      description
        "State parameters relating to interfaces associated
        with the instance";
      uses instance-interfaces-config;
      uses instance-interfaces-state;
    }
  }

  container connection-points {
    description
      "The set of connection points within a forwarding
      instance";

    list connection-point {
      key "connection-point-id";

      description
        "A connection point within a Layer 2 network instance.
        Each connection-point consists of a set of interfaces
        only one of which is active at any one time. Other than
        the specification of whether an interface is local
        (i.e., exists within this network-instance), or remote,
        all configuration and state parameters are common";

      leaf connection-point-id {
        type leafref {
          path "../config/connection-point-id";
        }
        description
          "A locally significant reference for the
          connection-point";
      }
```

```
            container config {
              description
                "Configuration parameters relating to a Layer 2
                network instance connection point";
              uses instance-connection-point-config;
            }
            container state {
              config false;
              description
                "Operational state parameters relating to a Layer 2
                network instance connection point";

              uses instance-connection-point-config;
              uses instance-connection-point-state;
            }

            container endpoints {
              when "../config/type = 'L2P2P' " +
                 "or ../config/type = 'L2VSI'" {
                description
                  "Configuration parameters to associate interfaces
                   into a common group for use in Layer 2 network
                   instances";
              }

              description
                "The set of endpoints which are grouped within the
                connection point";

              list endpoint {
                key "endpoint-id";

                description
                  "A list of the endpoints (interfaces or remote
                  connection points that can be used for this
                  connection point). The active endpoint is selected
                  based on the precedence that it is configured
                  with";

                leaf endpoint-id {
                  type leafref {
                    path "../config/endpoint-id";
                  }
                  description
                    "A pointer to the configured identifier for the
                    endpoint";
                }
```

```
            container config {
              description
                "Configuration parameters relating to the
                endpoint";
              uses instance-endpoint-config;
            }
            container state {
              config false;
              description
                "Operational state parameters relating to the
                endpoint";
              uses instance-endpoint-config;
              uses instance-endpoint-state;
            }
          }
        }
      }
    }

    container protocols {
      description
        "The routing protocols that are enabled for this
        network-instance.";

      list protocol {
        key "identifier name";

        description
          "A process (instance) of a routing protocol. Some
          systems may not support more than one instance of
          a particular routing protocol";

        leaf identifier {
          type leafref {
            path "../config/identifier";
          }
          description
            "The protocol name for the routing or forwarding
            protocol to be instantiated";
        }

        leaf name {
          type leafref {
            path "../config/name";
          }
          description
            "An operator-assigned identifier for the routing
            or forwarding protocol. For some processes this
```

```
            leaf may be system defined.";
          }

          container config {
            description
              "Configuration parameters relating to the routing
              protocol instance";

            uses protocols-config;
          }

          container state {
            config false;
            description
              "State parameters relating to the routing protocol
              instance";

            uses protocols-config;
            uses protocols-state;
          }

          container static {
            when "../config/identifier = 'STATIC'" {
              description
                "Include static route parameters only when the
                protocol is set to static";
            }
            description
              "Configuration and state parameters relating to
              static routes";
            uses lroute:local-static-top;
          }

          container aggregate {
            when "../config/identifier = 'LOCAL-AGGREGATE'" {
              description
                "Include aggregate route parameters only when the
                protocol is set to aggregate";
            }
            description
              "Configuration and state parameters relating to
              locally generated aggregate routes";
            uses lroute:local-aggregate-top;
          }
        }
      }
            }
      }
```

```
  }

  grouping instance-endpoint-config {
    description
      "Configuration data relating to an forwarding-instance
      endpoint";

    leaf endpoint-id {
      type string;
      description
        "An identifier for the endpoint";
    }

    uses instance-endpoint-local-remote;

    leaf precedence {
      type uint16;
      description
        "The precedence of the endpoint - the lowest precendence
        viable endpoint will be utilised as the active endpoint
        within a connection";
    }
  }

  grouping instance-endpoint-local-remote {
    description
      "A generic specification of a local or remote endpoint";

    choice local-remote {
      case local {
        leaf interface {
          type leafref {
            path "/network-instances/network-instance" +
                 "/interfaces/config/interface";
          }
          description
            "Reference to the local interface that is a member of
            the forwarding-instance";
        }
      }
      case remote {
        leaf neighbor {
          type inet:ip-address;
          description
            "The IP address of the device which hosts the
            remote end-point";
        }
```

```
        leaf virtual-circuit-identifier {
          type uint32;
          description
            "The virtual-circuit identifier that identifies the
            connection at the remote end-point";
        }
      }
      description
        "Configuration relating to an endpoint which can either be
        local (an interface), or remote. In the case where it is
        remote a neighbor IP address and virtual-circuit identifier
        must be specified";
    }
  }

  grouping instance-endpoint-state {
    description
      "Operational state data relating to a forwarding-instance
      endpoint";
    leaf active {
      type boolean;
      description
        "When the backup endpoint is active, the value of this
        parameter is set to true";
    }
  }

  grouping instance-connection-point-config {
    description
      "Configuration data relating to a forwarding-instance
      connection point";

    leaf connection-point-id {
      type string;
      description
        "An identifier for a connection point";
    }
  }

  grouping instance-connection-point-state {
    description
      "Operational state data relating to a forwarding-instance
      connection point";
  }

  grouping table-config {
    description
      "Configuration parameters relating to a L2/L2.5/L3 table that
```

```
      exists within the network instance";

    leaf table-name {
      type string;
      description
        "A human-readable name for the table";
    }

  }

  grouping table-state {
    description
      "State parameters relating to a table - this may be
      utilised to store generic structure for retrieving the contents
      of a RIB, FIB or LFIB";
    // placeholder
  }

  grouping inter-table-policies-config {
    description
      "Configuration entries that relate to how RIB or FIB entries
      are propagated between tables within the same network
      instance";

    leaf src-table {
      type leafref {
        // we are at table-connections/table-connection/config/.
        path "../../../../tables/table/table-name";
      }
      description
        "The source protocol for the table connection";
    }

    leaf dst-table {
      type leafref {
        path "../../../../tables/table/table-name";
      }
      description
        "The destination protocol for the table connection";
    }


  }

  grouping network-instance-config {
    description
      "Configuration parameters relating to a top-level network
      instance";
```

```
    leaf name {
      type string;
      description
        "An operator-assigned unique name for the forwarding
        instance";
    }

    leaf type {
      type identityref {
        base "nit:network-instance-type";
      }
      description
        "The type of network instance. The value of this leaf
        indicates the type of forwarding entries that should be
        supported by this network instance";
    }

    leaf enabled {
      type boolean;
      description
        "Whether the network instance should be configured to be
        active on the network element";
    }

    leaf description {
      type string;
      description
        "A free-form string to be used by the network operator to
        describe the function of this network instance";
    }

    leaf router-id {
      type yang:dotted-quad;
      description
        "A identifier for the local network instance - typically
        used within associated routing protocols or signalling
        routing information in another network instance";
    }

    leaf route-distinguisher {
      type nit:route-distinguisher;
      description
        "The route distinguisher that should be used for the local
        VRF or VSI instance when it is signalled via BGP.";
    }
  }

  grouping network-instance-state {
```

```
    description
      "Operational state parameters relating to a network instance";
  }

  grouping network-instance-l3vrf-config {
    description
      "Configuration parameters for a network instance of type
      l3vrf";

  }

  grouping protocols-config {
    description
      "Configuration parameters relating to a generic protocol
      instance within a network instance";

    leaf identifier {
      type identityref {
        base "pt:install-protocol-type";
      }
      description
        "The protocol identifier for the instance";
    }

    leaf name {
      type string;
      description
        "A unique name for the protocol instance";
    }

    leaf enabled {
      type boolean;
      description
        "A boolean value indicating whether the local protocol
        instance is enabled.";
    }

    leaf target-table {
     type leafref {
      path "../../../../tables/table/table-name";
     }
     description
      "The table (RIB, FIB, or LFIB) that the protocol should
      populate its entries in.";
    }
  }

  grouping protocols-state {
```

```
    description
      "Operational state parameters relating to a protocol instance";
  }

  grouping instance-interfaces-config {
    description
      "Base configuration parameters relating to the interfaces
      associated with a network instance";

    leaf-list interface {
      type leafref {
        path "/ocif:interfaces/ocif:interface/ocif:name";
      }
      description
        "Interfaces that are associated with the network instance";
    }
  }

  grouping instance-interfaces-state {
    description
      "State parameters relating to the interfaces associated with a
      network instance";
  }

  uses network-instance-top;

}

<CODE ENDS>


   <CODE BEGINS> file openconfig-network-instance-types.yang

   module openconfig-network-instance-types {

     yang-version "1";

     // namespace
     namespace "http://openconfig.net/yang/network-instance-types";

     prefix "nit";

     import openconfig-extensions { prefix "ocext"; }

     // meta
     organization "OpenConfig working group";

     contact
```

```
      "OpenConfig working group
      www.openconfig.net";

    description
      "Types associated with a network instance";

    ocext:openconfig-version "0.1.0";

    revision "2015-10-18" {
      description
        "Initial revision";
      reference "0.1.0";
    }

    // identity statements
    identity network-instance-type {
          description
           "A base identity which can be extended to indicate different
       types of network instance supported by a device.";
    }

    identity DEFAULT-INSTANCE {
      base network-instance-type;
      description
        "A special routing instance which acts as the 'default' or
        'global' routing instance for a network device.";
    }

    identity L3VRF {
      base network-instance-type;
      description
        "A private Layer 3 only routing instance which is formed of
        one or more RIBs";
    }

    identity L2VSI {
      base network-instance-type;
      description
        "A private Layer 2 only switch instance which is formed of
        one or more L2 forwarding tables";
    }

    identity L2P2P {
      base network-instance-type;
      description
        "A private Layer 2 only forwarding instance which acts as
        a point to point connection between two endpoints";
    }
```

```
    identity L2L3 {
      base network-instance-type;
      description
        "A private Layer 2 and Layer 2 forwarding instance";
    }

    // rjs note:
    // this should move to openconfig-types when merged
    typedef route-distinguisher {
      type union {
        // type 0: <2-byte administrator>:<4-byte assigned number>
        type string {
          pattern "(65[0-5][0-3][0-5]|[1-5][1-5][0-9][0-9][0-9]|"
                  + "[1-9]?[1-9]?[0-9][0-9]|[1-9]):"
                  + "(4[0-2][0-9][0-4][0-9][0-6][0-7][0-2][0-9][0-5]|"
                  + "[0-3][0-9]{9}|[1-9][0-9]{1,8}|[1-9])";
        }
        // type 1: <ip-address>:<2-byte assigned number>
        type string {
          pattern
            "(([0-9]|[1-9][0-9]|1[0-9][0-9]|2[0-4][0-9]|25[0-5])\.){3}"
            +  "([0-9]|[1-9][0-9]|1[0-9][0-9]|2[0-4][0-9]|25[0-5]):"
            +  "(65[0-5][0-3][0-5]|[1-5][1-5][0-9][0-9][0-9]|"
            +  "[1-9]?[1-9]?[0-9][0-9]|[1-9])";
        }
        // type 2: <4-byte as-number>:<2-byte assigned number>
        type string {
          pattern
            "(4[0-2][0-9][0-4][0-9][0-6][0-7][0-2][0-9][0-5]|"
            + "[0-3][0-9]{9}|[1-9][0-9]{1,8}|[1-9]):"
            + "65[0-5][0-3][0-5]|[1-5]{2}[0-9]{3}|"
            + "[1-9]{0,2}[0-9][0-9]|[1-9])";
        }
      }
      description "A route distinguisher value";
      reference "RFC4364";
    }

  }

  <CODE ENDS>
```

7. References

7.1. Normative references

[RFC6020] Bjorklund, M., Ed., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, DOI 10.17487/RFC6020, October 2010, <http://www.rfc-editor.org/info/rfc6020>.

[RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <http://www.rfc-editor.org/info/rfc3688>.

[I-D.openconfig-rtgwg-local-routing] Shaikh, A., George, J., Shakir, R., and F. Chen, "A Data Model for Locally Generated Routes in Service Provider Networks", draft-openconfig-rtgwg-local-routing-00 (work in progress), July 2015.

[I-D.ietf-rtgwg-policy-model] Shaikh, A., rjs@rob.sh, r., D'Souza, K., and C. Chase, "Routing Policy Configuration Model for Service Provider Networks", draft-ietf-rtgwg-policy-model-00 (work in progress), September 2015.

[I-D.ietf-idr-bgp-model] Shaikh, A., Shakir, R., Patel, K., Hares, S., D'Souza, K., Bansal, D., Clemm, A., Alex, A., Jethanandani, M., and X. Liu, "BGP Model for Service Provider Networks", draft-ietf-idr-bgp-model-00 (work in progress), July 2015.

7.2. Informative references

[YANG-REPO] "OpenConfig YANG Data Model Repository - Network Instance", November 2015, <https://github.com/openconfig/public/releases/network-instance>.

Appendix A. Acknowledgements

TODO

Author's Address

Rob Shakir
Jive Communications, Inc.
1275 West 1600 North, Suite 100
Orem, UT 84057

Email: rjs@rob.sh

Network Working Group
Internet-Draft
Intended status: Informational
Expires: January 27, 2017

A. Lindem, Ed.
Cisco Systems
L. Berger, Ed.
LabN Consulting, L.L.C.
D. Bogdanovic

C. Hopps
Deutsche Telekom
July 26, 2016

Network Device YANG Organizational Models
draft-rtgyangdt-rtgwg-device-model-05

Abstract

This document presents an approach for organizing YANG models in a comprehensive structure that may be used to configure and operate network devices. The structure is itself represented as a YANG model, with all of the related component models logically organized in a way that is operationally intuitive, but this model is not expected to be implemented. The identified component modules are expected to be defined and implemented on common network devices.

This document is derived from work submitted to the IETF by members of the informal OpenConfig working group of network operators and is a product of the Routing Area YANG Architecture design team.

Table of Contents

1. Introduction

"Operational Structure and Organization of YANG Models" [I-D.openconfig-netmod-model-structure], highlights the value of organizing individual, self-standing YANG [RFC6020] models into a more comprehensive structure. This document builds on that work and presents a derivative structure for use in representing the networking infrastructure aspects of physical and virtual devices. [I-D.openconfig-netmod-model-structure] and earlier versions of this document presented a single device-centric model root, this document no longer contains this element. Such an element would have translated to a single device management model that would be the root

of all other models and was judged to be overly restrictive in terms of definition, implementation, and operation.

The document presents a notional network device YANG organizational structure that provides a conceptual framework for the models that may be used to configure and operate network devices. The structure is itself presented as a YANG module, with all of the related component modules logically organized in a way that is operationally intuitive. This network device model is not expected to be implemented, but rather provide as context for the identified representative component modules with are expected to be defined, and supported on typical network devices.

This document refers to two new modules that are expected to be implemented. These models are defined to support the configuration and operation of network-devices that allow for the partitioning of resources from both, or either, management and networking perspectives. Two forms of resource partitioning are referenced:

The first form provides a logical partitioning of a network device where each partition is separately managed as essentially an independent network element which is 'hosted' by the base network device. These hosted network elements are referred to as logical network elements, or LNEs, and are supported by the logical-network-element module defined in [LNE-MODEL]. The module is used to identify LNEs and associate resources from the network-device with each LNE. LNEs themselves are represented in YANG as independent network devices; each accessed independently. Optionally, and when supported by the implementation, they may also be accessed from the host system. Examples of vendor terminology for an LNE include logical system or logical router, and virtual switch, chassis, or fabric.

The second form provides support what is commonly referred to as Virtual Routing and Forwarding (VRF) instances as well as Virtual Switch Instances (VSI), see [RFC4026]. In this form of resource partitioning multiple control plane and forwarding/bridging instances are provided by and managed via a single (physical or logical) network device. This form of resource partitioning is referred to as Network Instances and are supported by the network-instance module defined in [NI-MODEL]. Configuration and operation of each network-instance is always via the network device and the network-instance module.

This document was motivated by, and derived from, [I-D.openconfig-netmod-model-structure]. The requirements from that document have been combined with the requirements from "Consistent Modeling of Operational State Data in YANG",

[I-D.openconfig-netmod-opstate], into "NETMOD Operational State Requirements", [I-D.ietf-netmod-opstate-reqs]. This document is aimed at the requirement related to a common model-structure, currently Requirement 7, and also aims to provide a modeling base for Operational State representation.

The approach taken in this (and the original) document is to organize the models describing various aspects of network infrastructure, focusing on devices, their subsystems, and relevant protocols operating at the link and network layers. The proposal does not consider a common model for higher level network services. We focus on the set of models that are commonly used by network operators, and suggest a corresponding organization.

A significant portion of the text and model contained in this document was taken from the -00 of [I-D.openconfig-netmod-model-structure].

1.1. Status of Work and Open Issues

This version of the document and structure are a product of the Routing Area YANG Architecture design team and is very much a work in progress rather than a final proposal. This version is a major change from the prior version and this change was enabled by the work on the previously mentioned Schema Mount.

Schema Mount enables a dramatic simplification of the presented device model, particularly for "lower-end" devices which are unlikely to support multiple network instances or logical network elements. Should structural-mount/YSDL not be available, the more explicit tree structure presented in earlier versions of this document will need to be utilized.

The top open issues are:

1. This document will need to match the evolution and standardization of [I-D.openconfig-netmod-opstate] or [I-D.ietf-netmod-opstate-reqs] by the Netmod WG.

2. Interpretation of different policy containers requires clarification.

3. It may make sense to use the identityref structuring with hardware and QoS model.

4. Which document(s) define the base System management, network services, and oam protocols modules is TBD. This includes the

possibility of simply using RFC7317 in place of the presented System management module.

5. The model will be updated once the "opstate" requirements are addressed.

2. Module Overview

In this document, we consider network devices that support protocols and functions defined within the IETF Routing Area, e.g, routers, firewalls and hosts. Such devices may be physical or virtual, e.g., a classic router with custom hardware or one residing within a server-based virtual machine implementing a virtual network function (VNF). Each device may sub-divide their resources into logical network elements (LNEs) each of which provides a managed logical device. Examples of vendor terminology for an LNE include logical system or logical router, and virtual switch, chassis, or fabric. Each LNE may also support virtual routing and forwarding (VRF) and virtual switching instance (VSI) functions, which are referred to below as a network instances (NIs). This breakdown is represented in Figure 1.

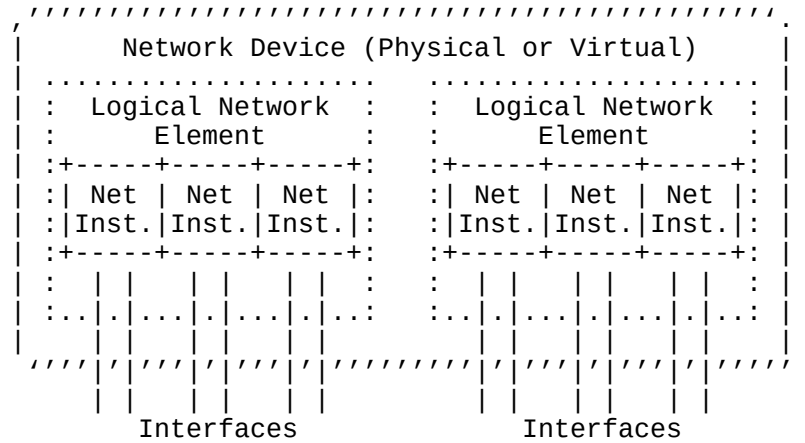


Figure 1: Module Element Relationships

A model for LNEs is described in [LNE-MODEL] and the model for network instances is covered in [NI-MODEL].

The presented notional network device module can itself be thought of as a "meta-model" as it describes the relationships between individual models. We choose to represent it also as a simple YANG module consisting of other models, which are in fact independent top

level individual models. Although it is never expected to be implemented.

The presented modules do not follow the hierarchy of any Particular implementation, and hence is vendor-neutral. Nevertheless, the structure should be familiar to network operators and also readily mapped to vendor implementations.

The overall structure is:

```
module: ietf-network-device
   +--rw modules-state             [I-D.ietf-netconf-yang-library]
   |
   +--rw interfaces                [RFC7223]
   +--rw hardware
   +--rw qos
   |
   +--rw system-management         [RFC7317 or derived]
   +--rw network-services
   +--rw oam-protocols
   |
   +--rw routing                   [I-D.ietf-netmod-routing-cfg]
   +--rw mpls
   +--rw ieee-dot1Q
   |
   +--rw acls                      [I-D.ietf-netmod-acl-model]
   +--rw key-chains                [I-D.ietf-rtgwg-yang-key-chain]
   |
   +--rw logical-network-elements [I-D.rtgyangdt-rtgwg-lne-model]
   +--rw network-instances         [I-D.rtgyangdt-rtgwg-ni-model]
```

The network device is composed of top level modules that can be used to configure and operate a network device. (This is a significant difference from earlier versions of this document where there was a strict model hierarchy.) Importantly the network device structure is the same for a physical network device or a logical network device, such as those instantiated via the logical-network-element model. Extra spacing is included to denote different types of modules included.

YANG library [I-D.ietf-netconf-yang-library] is included as it used to identify details of the top level modules supported by the (physical or logical) network device. Th ability to identify supported modules is particularly important for LNEs which may have a set of supported modules which differs from the set supported by the host network device.

The interface management model [RFC7223] is included at the top level. The hardware module is a placeholder for a future device-specific configuration and operational state data model. For example, a common structure for the hardware model might include chassis, line cards, and ports, but we leave this unspecified. The quality of service (QoS) section is also a placeholder module for device configuration and operational state data which relates to the treatment of traffic across the device. This document references augmentations to the interface module to support LNEs and NIs. Similar elements, although perhaps only for LNEs, may also need to be included as part of the definition of the future hardware and QoS modules.

System management, network services, and oam protocols represent new top level modules that are used to organize data models of similar functions. Additional information on each is provided below.

The routing and MPLS modules provide core support for the configuration and operation of a devices control plane and data plane functions. IEEE dot1Q [IEEE-8021Q] is an example of another module that provides similar functions for VLAN bridging, and other similar modules are also possible. Each of these modules is expected to be LNE and NI unaware, and to be instantiated as needed as part of the LNE and NI configuration and operation supported by the logical-network-element and network-instance modules. (Note that this is a change from [I-D.ietf-netmod-routing-cfg] which is currently defined with VRF/NI semantics.)

The access control list (ACL) and key chain modules are included as examples of other top level modules that may be supported by a network device.

The logical network element and network instance modules enable LNEs and NIs respectively and are defined below.

2.1. Interface Model Components

Interfaces are a crucial part of any network device’s configuration and operational state. They generally include a combination of raw physical interfaces, link-layer interfaces, addressing configuration, and logical interfaces that may not be tied to any physical interface. Several system services, and layer 2 and layer 3 protocols may also associate configuration or operational state data with different types of interfaces (these relationships are not shown for simplicity). The interface management model is defined by [RFC7223].

The logical-network-element and network-instance modules defined in [LNE-MODEL] and [NI-MODEL] augment the existing interface management model in two ways: The first, by the logical-network-element module, adds an identifier which is used on physical interface types to identify an associated LNE. The second, by the network-instance module, adds a name which is used on interface or sub-interface types to identify an associated network instance. Similarly, this name is also added for IPv4 and IPv6 types, as defined in [RFC7277].

The interface related augmentations are as follows:

```
module: ietf-logical-network-element
augment /if:interfaces/if:interface:
   +--rw bind-lne-name?   string

module: ietf-network-instance
augment /if:interfaces/if:interface:
   +--rw bind-network-instance-name?   string
augment /if:interfaces/if:interface/ip:ipv4:
   +--rw bind-network-instance-name?   string
augment /if:interfaces/if:interface/ip:ipv6:
   +--rw bind-network-instance-name?   string
```

The following is an example of envisioned combined usage. The interfaces container includes a number of commonly used components as examples:

```
            +--rw if:interfaces
            |  +--rw interface* [name]
            |     +--rw name                        string
            |     +--rw lne:bind-lne-name?          string
            |     +--rw ethernet
            |     |  +--rw ni:bind-network-instance-name? string
            |     |  +--rw aggregates
            |     |  +--rw rstp
            |     |  +--rw lldp
            |     |  +--rw ptp
            |     +--rw vlans
            |     +--rw tunnels
            |     +--rw ipv4
            |     |  +--rw ni:bind-network-instance-name? string
            |     |  +--rw arp
            |     |  +--rw icmp
            |     |  +--rw vrrp
            |     |  +--rw dhcp-client
            |     +--rw ipv6
            |        +--rw ni:bind-network-instance-name? string
            |        +--rw vrrp
            |        +--rw icmpv6
            |        +--rw nd
            |        +--rw dhcpv6-client
```

The [RFC7223] defined interface model is structured to include all interfaces in a flat list, without regard to logical or virtual instances (e.g., VRFs) supported on the device. The bind-lne-name and bind-network-instance-name leaves provide the association between an interface and its associated LNE and NI (e.g., VRF or VSI).

2.2. System Management

[Editor’s note: need to discuss and resolve relationship between this structure and RFC7317 and determine if 7317 is close enough to simply use as is.]

System management is expected to reuse definitions contained in [RFC7317]. It is expected to be instantiated per device and LNE. Its structure is shown below:

```
    module: ietf-network-device
       +--rw system-management
       |  +--rw system-management-global
       |  +--rw system-management-protocol* [type]
       |     +--rw type    identityref
```

System-management-global is used for configuration information and state that is independent of a particular management protocol. System-management-protocol is a list of management protocol specific elements. The type-specific sub-modules are expected to be defined.

The following is an example of envisioned usage:

```
module: ietf-network-device
   +--rw system-management
      +--rw system-management-global
      |  +--rw statistics-collection
      |  ...
      +--rw system-management-protocol* [type]
      |  +--rw type=syslog
      |  +--rw type=dns
      |  +--rw type=ntp
      |  +--rw type=ssh
      |  +--rw type=tacacs
      |  +--rw type=snmp
      |  +--rw type=netconf
```

2.3. Network Services

A device may provide different network services to other devices, for example a device my act as a DHCP server. The model may be instantiated per device, LNE, and NI. An identityref is used to identify the type of specific service being provided and its associated configuration and state information. The defined structure is as follows:

```
module: ietf-network-device
   +--rw network-services
   |  +--rw network-service* [type]
   |     +--rw type    identityref
```

The following is an example of envisioned usage: Examples shown below include a device-based Network Time Protocol (NTP) server, a Domain Name System (DNS) server, and a Dynamic Host Configuration Protocol (DHCP) server:

```
module: ietf-network-device
   +--rw network-services
      +--rw network-service* [type]
         +--rw type=ntp-server
         +--rw type=dns-server
         +--rw type=dhcp-server
```

2.4. OAM Protocols

OAM protocols that may run within the context of a device are grouped within the oam-protocols model. The model may be instantiated per device, LNE, and NI. An identifyref is used to identify the information and state that may relate to a specific OAM protocol. The defined structure is as follows:

```
module: ietf-network-device
   +--rw oam-protocols
      +--rw oam-protocol* [type]
         +--rw type    identityref
```

The following is an example of envisioned usage. Examples shown below include Bi-directional Forwarding Detection (BFD), Ethernet Connectivity Fault Management (CFM), and Two-Way Active Measurement Protocol (TWAMP):

```
module: ietf-network-device
   +--rw oam-protocols
      +--rw oam-protocol* [type]
         +--rw type=bfd
         +--rw type=cfm
         +--rw type=twamp
```

2.5. Routing

Routing protocol and IP forwarding configuration and operation information is modeled via a routing model, such as the one defined in [I-D.ietf-netmod-routing-cfg].

The routing module is expected to include all IETF defined control plane protocols, such as BGP, OSPF, LDP and RSVP-TE. It is also expected to support configuration and operation of or more routing information bases (RIB). A RIB is a list of routes complemented with administrative data. Finally, policy is expected to be represented within each control plane protocol and RIB.

The anticipated structure is as follows:

```
module: ietf-network-device
    +--rw rt:routing                [I-D.ietf-netmod-routing-cfg]
       +--rw control-plane-protocol* [type]
       |  +--rw type      identityref
       |  +--rw policy
       +--rw rib* [name]
          +--rw name           string
          +--rw description?   string
          +--rw policy
```

2.6. MPLS

MPLS data plane related information is grouped together, as with the previously discussed modules, is unaware of VRFs/NIs. The model may be instantiated per device, LNE, and NI. MPLS control plane protocols are expected to be included in Section 2.5. MPLS may reuse and build on [I-D.openconfig-mpls-consolidated-model] or other emerging models and has an anticipated structure as follows:

```
module: ietf-network-device
     +--rw mpls
        +--rw global
        +--rw lsps* [type]
           +--rw type    identityref
```

Type refers to LSP type, such as static, traffic engineered or routing congruent. The following is an example of such usage:

```
module: ietf-network-device
     +--rw mpls
        +--rw global
        +--rw lsps* [type]
            +--rw type=static
            +--rw type=constrained-paths
            +--rw type=igp-congruent
```

3. Security Considerations

The network-device model structure described in this document does not define actual configuration and state data, hence it is not directly responsible for security risks.

Each of the component models that provide the corresponding configuration and state data should be considered sensitive from a security standpoint since they generally manipulate aspects of network configurations. Each component model should be carefully evaluated to determine its security risks, along with mitigations to reduce such risks.

LNE portion is TBD

NI portion is TBD

4. IANA Considerations

This YANG model currently uses a temporary ad-hoc namespace. If it is placed or redirected for the standards track, an appropriate namespace URI will be registered in the "IETF XML Registry" [RFC3688]. The YANG structure modules will be registered in the "YANG Module Names" registry [RFC6020].

5. Network Device Model Structure

```
<CODE BEGINS> file "ietf-network-device@2016-05-01.yang"
module ietf-network-device {

  yang-version "1";

  // namespace
  namespace "urn:ietf:params:xml:ns:yang:ietf-network-device";

  prefix "nd";

  // import some basic types

  // meta
  organization "IETF RTG YANG Design Team Collaboration
                with OpenConfig";

  contact
      "Routing Area YANG Architecture Design Team -
       <rtg-dt-yang-arch@ietf.org>";

  description
    "This module describes a model structure for YANG
     configuration and operational state data models. Its intent is
     to describe how individual device protocol and feature models
     fit together and interact.";

  revision "2016-05-01" {
    description
      "IETF Routing YANG Design Team Meta-Model";
    reference "TBD";
  }

  // extension statements
```

```
     // identity statements

     identity oam-protocol-type {
         description
             "Base identity for derivation of OAM protocols";
     }

     identity network-service-type {
         description
             "Base identity for derivation of network services";
     }

      identity system-management-protocol-type {
         description
             "Base identity for derivation of system management
              protocols";
      }

      identity oam-service-type {
         description
             "Base identity for derivation of Operations,
              Administration, and Maintenance (OAM) services.";
      }

      identity control-plane-protocol-type {
         description
             "Base identity for derivation of control-plane protocols";
      }

      identity mpls-lsp-type {
         description
             "Base identity for derivation of MPLS LSP typs";
      }

     // typedef statements

     // grouping statements

     grouping ribs {
       description
         "Routing Information Bases (RIBs) supported by a
          network-instance";
       container ribs {
           description
               "RIBs supported by a network-instance";
           list rib {
               key "name";
               min-elements "1";
```

```
            description
                "Each entry represents a RIB identified by the
               'name' key. All routes in a RIB must belong to the
                same address family.

                For each routing instance, an implementation should
                provide one system-controlled default RIB for each
                supported address family.";
            leaf name {
                type string;
                description
                    "The name of the RIB.";
            }
            reference "draft-ietf-netmod-routing-cfg";
            leaf description {
                type string;
                description
                    "Description of the RIB";
            }
            // Note that there is no list of interfaces within
            container policy {
                description "Policy specific to RIB";
            }
        }
      }
    }

    // top level device definition statements
    container ietf-yang-library {
      description
        "YANG Module Library as defined in
         draft-ietf-netconf-yang-library";
    }

    container interfaces {
      description
       "Interface list as defined by RFC7223/RFC7224";
    }

    container hardware {
      description
        "Hardware / vendor-specific data relevant to the platform.
        This container is an anchor point for platform-specific
        configuration and operational state data.  It may be further
        organized into chassis, line cards, ports, etc.  It is
        expected that vendor or platform-specific augmentations
        would be used to populate this part of the device model";
    }
```

```
container qos {
  description "QoS features, for example policing, shaping, etc.";
}

container system-management {
    description
      "System management for physical or virtual device.";
    container system-management-global {
        description "System management - with reuse of RFC 7317";
    }
    list system-management-protocol {
        key "type";
        leaf type {
            type identityref {
                base system-management-protocol-type;
            }
            mandatory true;
            description
                "Syslog, ssh, TACAC+, SNMP, NETCONF, etc.";
        }
        description "List of system management protocol
                     configured for a logical network
                     element.";
    }
}

container network-services {
    description
        "Container for list of configured network
         services.";
    list network-service {
        key "type";
        description
            "List of network services configured for a
             network instance.";
        leaf type {
            type identityref {
                base network-service-type;
            }
            mandatory true;
            description
                "The network service type supported within
                 a network instance, e.g., NTP server, DNS
                 server, DHCP server, etc.";
        }
    }
}
```

```
     container oam-protocols {
         description
             "Container for configured OAM protocols.";
         list oam-protocol {
             key "type";
             leaf type {
                 type identityref {
                     base oam-protocol-type;
                 }
                 mandatory true;
                 description
                     "The Operations, Administration, and
                      Maintenance (OAM) protocol type, e.g., BFD,
                      TWAMP, CFM, etc.";
             }
             description
                 "List of configured OAM protocols.";
         }
     }

     container routing {
       description
         "The YANG Data Model for Routing Management revised to be
          Network Instance / VRF independent. ";
       // Note that there is no routing or network instance
       list control-plane-protocol {
           key "type";
           description
               "List of control plane protocols configured for
                a network instance.";
           leaf type {
               type identityref {
                   base control-plane-protocol-type;
               }
               description
                   "The control plane protocol type, e.g., BGP,
                    OSPF IS-IS, etc";
           }
           container policy {
               description
                   "Protocol specific policy,
                   reusing [RTG-POLICY]";
           }
       }
       list rib {
           key "name";
           min-elements "1";
           description
```

```
                "Each entry represents a RIB identified by the
               'name' key. All routes in a RIB must belong to the
                same address family.

                For each routing instance, an implementation should
                provide one system-controlled default RIB for each
                supported address family.";
            leaf name {
                type string;
                description
                    "The name of the RIB.";
            }
            reference "draft-ietf-netmod-routing-cfg";
            leaf description {
                type string;
                description
                    "Description of the RIB";
            }
            // Note that there is no list of interfaces within
            container policy {
                description "Policy specific to RIB";
            }
          }
        }

        container mpls {
            description "MPLS and TE configuration";
            container global {
                description "Global MPLS configuration";
            }
            list lsps {
                key "type";
                description
                    "List of LSP types.";
                leaf type {
                    type identityref {
                        base mpls-lsp-type;
                    }
                    mandatory true;
                    description
                        "MPLS and Traffic Engineering protocol LSP types,
                         static, LDP/SR (igp-congruent),
                         RSVP TE (constrained-paths) , etc.";
                }
            }
        }

        container ieee-dot1Q {
```

```
      description
        "The YANG Data Model for VLAN bridges as defined by the IEEE";
    }

    container ietf-acl {
      description "Packet Access Control Lists (ACLs) as specified
                     in draft-ietf-netmod-acl-model";
    }

    container ietf-key-chain {
      description "Key chains as specified in
                   draft-ietf-rtgwg-yang-key-chain;";
    }

    container logical-network-element {
      description
        "This module is used to support multiple logical network
         elements on a single physical or virtual system.";
    }

    container network-instance {
      description
        "This module is used to support multiple network instances
         within a single physical or virtual device.  Network
         instances are commonly know as VRFs (virtual routing
         and forwarding) and VSIs (virtual switching instances).";
    }
    // rpc statements

    // notification statements

  }
  <CODE ENDS>
```

6. References

6.1. Normative References

[LNE-MODEL]
Berger, L., Hopps, C., Lindem, A., and D. Bogdanovic, "Logical Network Element Model", draft-rtgyangdt-rtgwg-lne-model-00.txt (work in progress), May 2016.

[NI-MODEL]
Berger, L., Hopps, C., Lindem, A., and D. Bogdanovic, "Network Instance Model", draft-rtgyangdt-rtgwg-ni-model-00.txt (work in progress), May 2016.

[RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <http://www.rfc-editor.org/info/rfc3688>.

[RFC4026] Andersson, L. and T. Madsen, "Provider Provisioned Virtual Private Network (VPN) Terminology", RFC 4026, DOI 10.17487/RFC4026, March 2005, <http://www.rfc-editor.org/info/rfc4026>.

[RFC6020] Bjorklund, M., Ed., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, DOI 10.17487/RFC6020, October 2010, <http://www.rfc-editor.org/info/rfc6020>.

[RFC7223] Bjorklund, M., "A YANG Data Model for Interface Management", RFC 7223, DOI 10.17487/RFC7223, May 2014, <http://www.rfc-editor.org/info/rfc7223>.

[RFC7277] Bjorklund, M., "A YANG Data Model for IP Management", RFC 7277, DOI 10.17487/RFC7277, June 2014, <http://www.rfc-editor.org/info/rfc7277>.

[RFC7317] Bierman, A. and M. Bjorklund, "A YANG Data Model for System Management", RFC 7317, DOI 10.17487/RFC7317, August 2014, <http://www.rfc-editor.org/info/rfc7317>.

6.2. Informative References

[I-D.ietf-netconf-yang-library]
Bierman, A., Bjorklund, M., and K. Watsen, "YANG Module Library", draft-ietf-netconf-yang-library-05 (work in progress), April 2016.

[I-D.ietf-netmod-opstate-reqs]
Watsen, K. and T. Nadeau, "Terminology and Requirements for Enhanced Handling of Operational State", draft-ietf-netmod-opstate-reqs-03 (work in progress), January 2016.

[I-D.ietf-netmod-routing-cfg]
Lhotka, L. and A. Lindem, "A YANG Data Model for Routing Management", draft-ietf-netmod-routing-cfg-20 (work in progress), October 2015.

[I-D.openconfig-mpls-consolidated-model]
George, J., Fang, L., eric.osborne@level3.com, e., and R. Shakir, "MPLS / TE Model for Service Provider Networks", draft-openconfig-mpls-consolidated-model-02 (work in progress), October 2015.

[I-D.openconfig-netmod-model-structure]
Shaikh, A., Shakir, R., D'Souza, K., and L. Fang, "Operational Structure and Organization of YANG Models", draft-openconfig-netmod-model-structure-00 (work in progress), March 2015.

[I-D.openconfig-netmod-opstate]
Shakir, R., Shaikh, A., and M. Hines, "Consistent Modeling of Operational State Data in YANG", draft-openconfig-netmod-opstate-01 (work in progress), July 2015.

[IEEE-8021Q]
Holness, M., "IEEE 802.1Q YANG Module Specifications", IEEE-Draft http://www.ieee802.org/1/files/public/docs2015/new-mholness-yang-8021Q-0515-v04.pdf, May 2015.

Appendix A. Acknowledgments

This document is derived from draft-openconfig-netmod-model-structure-00. We thank the Authors of that document and acknowledge their indirect contribution to this work. The authors include: Anees Shaikh, Rob Shakir, Kevin D'Souza, Luyuan Fang, Qin Wu, Stephane Litkowski and Gang Yan.

This work was discussed in and produced by the Routing Area Yang Architecture design team. Members at the time of writing included Acee Lindem, Anees Shaikh, Christian Hopps, Dean Bogdanovic, Lou Berger, Qin Wu, Rob Shakir, Stephane Litkowski, and Gang Yan.

The identityref approach was proposed by Mahesh Jethanandani.

The RFC text was produced using Marshall Rose's xml2rfc tool.

Authors' Addresses

Acee Lindem (editor)
Cisco Systems
301 Midenhall Way
Cary, NC 27513
USA

Email: acee@cisco.com

Lou Berger (editor)
LabN Consulting, L.L.C.

Email: lberger@labn.net

Dean Bogdanovic

Email: ivandean@gmail.com

Christan Hopps
Deutsche Telekom

Email: chopps@chopps.org

IDR Working Group
Internet-Draft
Intended status: Standards Track
Expires: January 9, 2017

G. Van de Velde, Ed.
W. Henderickx
Nokia
K. Patel
A. Sreekantiah
Cisco Systems
Z. Li
S. Zhuang
N. Wu
Huawei Technologies
July 8, 2016

Flowspec Indirection-id Redirect
draft-vandevelde-idr-flowspec-path-redirect-03

Abstract

Flowspec is an extension to BGP that allows for the dissemination of traffic flow specification rules. This has many possible applications but the primary one for many network operators is the distribution of traffic filtering actions for DDoS mitigation. The flow-spec standard RFC5575 [2] defines a redirect-to-VRF action for policy-based forwarding but this mechanism is not always sufficient, particularly if the redirected traffic needs to be steered into an engineered path or into a service plane.

This document defines a new extended community known as redirect-to-indirection-id (32-bit) flowspec action to provide advanced redirection capabilities on flowspec clients. When activated, the flowspec extended community is used by a flowspec client to find the correct next-hop entry within a localised indirection-id mapping table.

The functionality present in this draft allows a network controller to decouple flowspec functionality from the creation and maintainance of the network’s service plane itself including the setup of tunnels and other service constructs that could be managed by other network devices.

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 [1].

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

Flowspec RFC5575 [2] is an extension to BGP that allows for the dissemination of traffic flow specification rules. This has many possible applications, however the primary one for many network operators is the distribution of traffic filtering actions for DDoS mitigation.

Every flowspec policy route is effectively a rule, consisting of a matching part (encoded in the NLRI field) and an action part (encoded in one or more BGP extended communities). The flow-spec standard RFC5575 [2] defines widely-used filter actions such as discard and rate limit; it also defines a redirect-to-VRF action for policy-based forwarding. Using the redirect-to-VRF action to steer traffic towards an alternate destination is useful for DDoS mitigation but using this technology can be cumbersome when there is need to steer the traffic onto an engineered traffic path.

This draft proposes a new redirect-to-indirection-id flowspec action facilitating an anchor point for policy-based forwarding onto an engineered path or into a service plane. The flowspec client consuming and utilizing the new flowspec indirection-id extended-community finds the redirection information within a localised indirection-id mapping table. The localised mapping table is a table construct providing at one side the table key and at the other side next-hop information. The table key consists out the combination of indirection-id type and indirection-id 32-bit value.

The redirect-to-indirection-id flowspec action is encoded in a newly defined BGP extended community. In addition, the type of redirection can be configured as an extended community indirection-id type field.

This draft defines the indirection-id extended-community and the wellknown indirection-id types. The specific solution to construct the localised indirection-id mapping table are out-of-scope of this document.

2. indirection-id and indirection-id table

An indirection-id is an abstract number (32-bit value) used as identifier for a localised indirection decision. The indirection-id will allow a flowspec client to redirect traffic into a service plane or onto an engineered traffic path. e.g. When a BGP flowspec controller signals a flowspec client the indirection-id extended community, then the flowspec client uses the indirection-id to make a

recursive lookup to retrieve next-hop information found in a localised indirection mapping table.

The indirection-id table is a router localised table. The indirection-id table is constructed out of table keys mapped to flowspec client localised redirection information. The table key is created by the combination of the indirection-id type and the indirection-id 32-bit value. Each entry in the indirection-table key maps to sufficient information (parameters regarding encapsulation, interface, QoS, etc...) to successfully redirect traffic.

3. Use Case Scenarios

This section describes use-case scenarios when deploying redirect-to-indirection-id.

3.1. Redirection shortest Path tunnel

A first use-case is allowing a BGP Flowspec controller to send a single flowspec policy route (i.e. flowspec_route#1) to many BGP flowspec clients. This flowspec route signals the Flowspec clients to redirect traffic onto a tunnel towards a single IP destination address.

For this first use-case scenario, the flowspec client receives from the flowspec controller a flowspec route (i.e. flowspec_route#1) including the redirect-to-indirection-id extended community. The redirect-to-indirection-id extended community contains the key (indirection-id type + indirection-id 32-bit value) to select the corresponding next-hop information from the flowpsec client localised indirection-id table. The resulting next-hop information for this use-case is a remote tunnel end-point IP address with accordingly sufficient tunnel encapsulation information to forward the packet accordingly.

For redirect to shortest path tunnel it is required that the tunnel MUST be operational and allow packets to be exchanged between tunnel head- and tail-end.

3.2. Redirection to path-engineered tunnels

For a second use-case, it is expected that the flowspec client redirect traffic matches the flowspec rule, onto a path engineered tunnel. The path engineered tunnel on the flowspec client SHOULD be created by out-of-band mechanisms. Each path engineered tunnel deployed for flowspec redirection, has a unque key as an identifier. consequently, the key (=indirection-id type and indirection-id 32-bit value) uniquely identifies a single path engineered tunnel on the

flowspec client. The localised indirection-id mapping table is the collection of all keys corresponding all path engineered tunnels on the flowspec client.

For this second use-case scenario, the flowspec controller sends a flowspec route (i.e. flowspec_route#2) to the flowspec clients. The flowspec clients, respectively receive the flowspec route. The redirect-to-indirection-id extended community contains the key (indirection type + indirection-id 32-bit value) to select the corresponding next-hop information from the flowpsec client localised indirection-id table. The resulting next-hop information for this use-case is path engineered tunnel information and has sufficient tunnel encapsulation information to forward the packet according the expectations of the flowspec controller.

A concrete example of this use-case can be found in segment routed networks where path engineered tunnels can be setup by means of a controller signaling explicit paths to peering routers. In such a case, the indirection-id references to a Segment Routing Binding SID, while the indirection-id type references the Bindging SID semantic. The Binding SID is a segment identifier value (as per segment routing definitions in [I-D.draft-ietf-spring-segment-routing] [6]) used to associate with an explicit path and can be setup by a controller using BGP as specified in [I-D.sreekantiah-idr-segment-routing-te] [5] or using PCE as detailed in draft-ietf-pce-segment-routing [7]. When a BGP speaker receives a flow-spec route with a ’redirect to Binding SID’ extended community, it installs a traffic filtering rule that matches the packets described by the NLRI field and redirects them to the explicit path associated with the Binding SID. The explicit path is specified as a set/stack of segment identifiers as detailed in the previous documents. The stack of segment identifiers is now imposed on packets matching the flow-spec rule to perform redirection as per the explicit path setup prior. The encoding of the Binding SID value is specified in section 4, with the indirection-id field now encoding the associated value for the binding SID.

3.3. Redirection to Next-next-hop tunnels

A Third use-case is when a BGP Flowspec controller sends a single flowspec policy route to flowpsec clients to signal redirection towards next-next-hop tunnels. In this use-case The flowspec rule is instructing the Flowspec client to redirect traffic using a sequence of indirection-id extended communities. The sequence of indirection-ids is managed using Tunnel IDs (TID). i.e. a classic example would be DDoS mitigation towards Segment Routing Central Egress Path Engineering [4]. To steer DDoS traffic towards egress peer engineering paths, a first indirection-id will steer traffic onto a

tunnel to an egress router, while a second indirection-id is used steer the traffic at this egress router onto a particular interface or towards a peer. The flowspec client will for this use-case dynamically append all segment routing segments to steer the DDoS traffic through the EPE path.

To achieve this type of redirection to next-next-hop tunnels, multiple indirection-ids, each using a unique Tunnel ID are imposed upon a the flowspec policy rule. The Tunnel ID will allow the flowspec client to sequence the indirection-ids for correct next-next-hop tunnel constructs.

4. Redirect to indirection-id Community

This document defines a new BGP extended community known as a Redirect-to-indirection-id extended community. This extended community is a new transitive extended community with the Type and the Sub-Type field to be assigned by IANA. The format of this extended community is show in Figure 1.

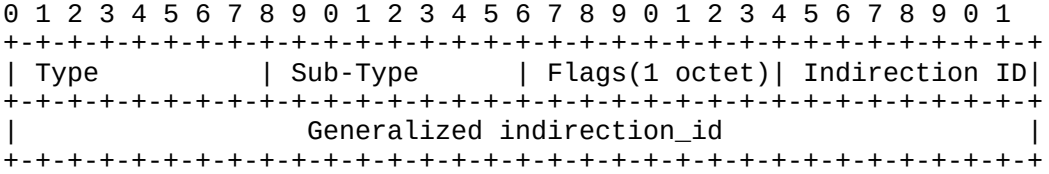


Figure 1

The meaning of the extended community fields are as follows:

Type: 1 octet to be assigned by IANA.

Sub-Type: 1 octet to be assigned by IANA.

Flags: 1 octet field. Following Flags are defined.

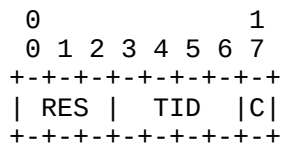


Figure 2

The least-significant Flag bit is defined as the 'C' (or copy) bit. When the 'C' bit is set the redirection applies to copies of the matching packets and not to the original traffic stream.

The 'TID' field identifies a 4 bit Table-id field. This field is used to provide the flowspec client an indication how and where to sequence the received indirection-ids to redirecting traffic. TID value 0 indicates that Table-id field is NOT set and SHOULD be ignored.

All bits other than the 'C' and 'TID' bits MUST be set to 0 by the originating BGP speaker and ignored by receiving BGP speakers.

Indirection ID: 1 octet value. This draft defines following indirection_id Types:

0 - Localised ID

1 - Node ID

2 - Agency ID

3 - AS (Autonomous System) ID

4 - Anycast ID

5 - Multicast ID

6 - Tunnel ID (Tunnel Binding ID)

7 - VPN ID

8 - OAM ID

9 - ECMP (Equal Cost Multi-Path) ID

10 - QoS ID

11 - Bandwidth-Guarantee ID

12 - Security ID

13 - Multi-Topology ID

5. Redirect using localised indirection-id mapping table

When a BGP speaker receives a flowspec policy route with a ’redirect to indirection-id’ extended community and this route represents the one and only best path or an equal cost multipath, it installs a traffic filtering rule that matches the packets described by the NLRI field and redirects them (C=0) or copies them (C=1) towards the indirection-id local recursed path. To construct the local recursed path, the flowspec client does a local indirection-id mapping table lookup using the key comprised of the indirection-id 32-bit value and indirection-id type to retrieve the correct redirection information.

6. Validation Procedures

The validation check described in RFC5575 [2] and revised in [3] SHOULD be applied by default to received flow-spec routes with a ’redirect to indirection-id’ extended community. This means that a flow-spec route with a destination prefix subcomponent SHOULD NOT be accepted from an EBGP peer unless that peer also advertised the best path for the matching unicast route.

It is possible from a semenatics perspective to have multiple redirect actions defined within a single flowspec rule. When a BGP flowspec NLRI has a ’redirect to indirection-id’ extended community attached resulting in valid redirection then it MUST take priority above all other redirect actions emposed. However, if the ’redirect to indirection-id’ does not result in a valid redirection, then the flowspec rule must be processed as if the ’redirect to indirection-id’ community was not attached to the flowspec route and MUST provide an indication within the BGP routing table that the respective ’redirect to indirection-id’ resulted in an invalid redirection action.

7. Security Considerations

A system using ’redirect-to-indirection-id’ extended community can cause during the redirect mitigation of a DDoS attack result in overflow of traffic received by the mitigation infrastructure.

8. Acknowledgements

This document received valuable comments and input from IDR working group including Adam Simpson, Mustapha Aissaoui, Jan Mertens, Robert Raszuk, Jeff Haas, Susan Hares and Lucy Yong

9. IANA Considerations

This document requests a new type and sub-type for the Redirect to indirection-id Extended community from the "Transitive Extended community" registry. The Type name shall be "Redirect to indirection-id Extended Community" and the Sub-type name shall be 'Flow-spec Redirect to 32-bit Path-id'.

In addition, this document requests IANA to create a new registry for Redirect to indirection-id Extended Community INDIRECTION-IDs as follows:

Under "Transitive Extended Community:"

Registry: "Redirect Extended Community indirection_id"

Reference: [RFC-To-Be]

Registration Procedure(s): First Come, First Served

Registry: "Redirect Extended Community indirection_id"

Value	Code	Reference
0	Localised ID	[RFC-To-Be]
1	Node ID	[RFC-To-Be]
2	Agency ID	[RFC-To-Be]
3	AS (Autonomous System) ID	[RFC-To-Be]
4	Anycast ID	[RFC-To-Be]
5	Multicast ID	[RFC-To-Be]
6	Tunnel ID (Tunnel Binding ID)	[RFC-To-Be]
7	VPN ID	[RFC-To-Be]
8	OAM ID	[RFC-To-Be]
9	ECMP (Equal Cost Multi-Path) ID	[RFC-To-Be]
10	QoS ID	[RFC-To-Be]
11	Bandwidth-Guarantee ID	[RFC-To-Be]
12	Security ID	[RFC-To-Be]
13	Multi-Topology ID	[RFC-To-Be]

Figure 3

10. References

10.1. Normative References

[1] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997, <http://xml.resource.org/public/rfc/html/rfc2119.html>.

[2] Marques, P., Sheth, N., Raszuk, R., Greene, B., Mauch, J., and D. McPherson, "Dissemination of Flow Specification Rules", RFC 5575, DOI 10.17487/RFC5575, August 2009, <http://www.rfc-editor.org/info/rfc5575>.

10.2. Informative References

[3] Uttaro, J., Filsfils, C., Alcaide, J., and P. Mohapatra, "Revised Validation Procedure for BGP Flow Specifications", January 2014.

[4] Filsfils, C., Previdi, S., Aries, E., Ginsburg, D., and D. Afanasiev, "Segment Routing Centralized Egress Peer Engineering", October 2015.

[5] Sreekantiah, A., Filsfils, C., Previdi, S., Sivabalan, S., Mattes, P., and S. Lin, "Segment Routing Traffic Engineering Policy using BGP", October 2015.

[6] Filsfils, C., Previdi, S., Decraene, B., Litkowski, S., Shakir, R., Bashandy, A., Horneffer, M., Henderickx, W., Tantsura, J., Crabbe, E., Milojevic, I., and S. Ytti, "Segment Routing Architecture", December 2015.

[7] Sivabalan, S., Medved, M., Filsfils, C., Litkowski, S., Raszuk, R., Bashandy, A., Lopez, V., Tantsura, J., Henderickx, W., Hardwick, J., Milojevic, I., and S. Ytti, "PCEP Extensions for Segment Routing", December 2015.

Authors' Addresses

Gunter Van de Velde (editor)
Nokia
Antwerp
BE

Email: gunter.van_de_velde@nokia.com

Wim Henderickx
Nokia
Antwerp
BE

Email: wim.henderickx@nokia.com

Keyur Patel
Cisco Systems
170 W. Tasman Drive
San Jose, CA 95134
USA

Email: keyupate@cisco.com

Arjun Sreekantiah
Cisco Systems
170 W. Tasman Drive
San Jose, CA 95134
USA

Email: asreekan@cisco.com

Zhenbin Li
Huawei Technologies
Huawei Bld., No. 156 Beiquing Rd
Beijing 100095
China

Email: lizhenbin@huawei.com

Shunwan Zhuang
Huawei Technologies
Huawei Bld., No. 156 Beiquing Rd
Beijing 100095
China

Email: zhuangshunwan@huawei.com

Nan Wu
Huawei Technologies
Huawei Bld., No. 156 Beiquing Rd
Beijing 100095
China

Email: eric.wu@huawei.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: January 4, 2016

A. Wang
China Telecom
Z. Wang
Y. Zhuang
Huawei

July 3, 2015

YANG Data Model for Tunnel Management
draft-wwz-netmod-yang-tunnel-cfg-00

Abstract

This document defines a YANG data model for the configuration and management of generic tunnels. The data model includes configuration data and state data. And it can serve as a base model which is augmented with technology-specific details in other, more specific tunnel models.

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 January 4, 2016.

Table of Contents

1. Introduction

Tunneling mechanisms provide isolated communication from one VPN edge device to another in the VPN solutions. Available tunneling mechanisms include (but are not limited to): GRE [RFC2784] [RFC2890], IP-in-IP encapsulation [RFC2003] [RFC2473], IPsec [RFC2401] [RFC2402], and MPLS [RFC3031] [RFC3035], VXLAN ([RFC7348]), VXLAN-GPE ([VXLAN-GPE]), NVGRE ([NVGRE]), GTP [GTP-U], and MPLS-in-GRE ([RFC2784], [RFC2890], [RFC4023]).

[RFC4110] discusses some common characteristics shared by all forms of tunneling, and some common problems to which tunnels provide a solution from the following perspectives:

- Multiplexing
- QoS/SLA
- Tunnel setup and maintenance
- Security

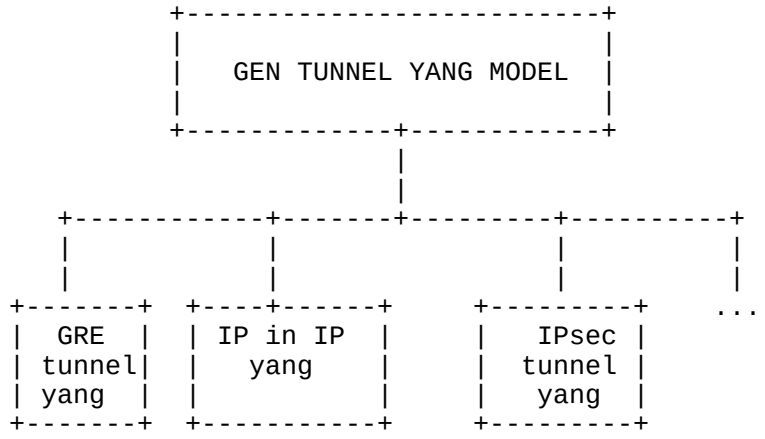
This document defines a yang data model[RFC6020] to represent common building block for tunnels configuration data and state data. This model can be augmented with technology specifics of particular types of tunnel.

2. Conventions and Terminology

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].

3. Architecture of Generic YANG Model for tunnel

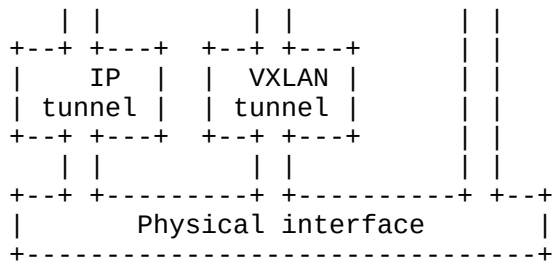
In this document we define a common core tunnel model. The YANG model defined here is generic such that other technologies can extend it for technology specific needs. The Generic Tunnel YANG model acts as the root for other Tunnel YANG models. This allows users to span across Tunnel of different technologies through a uniform API set. Figure 1 depicts the relationship of different Tunnel YANG models to the Generic Tunnel YANG Model. Some technologies may have different sub-technologies. As an example, consider Network IP-in-IP Tunnel. These could employ IPv4-in-IPv4, IPv6-in-IPv4, and other methods as encapsulation protocol. Figure 1 depicts relationship of different YANG modules.



Relationship of technology specific TUNNEL YANG model to generic (base) YANG model

3.1. Relationship to interface yang data model

This section clarifies the relationship of this yang module to the interfaces yang [RFC7223]. Tunnels are handled by creating a logical interface for each tunnel. Each logical interface (physical or virtual) need to map to interface yang model [RFC7223]. To do so, in this draft, we augment interface yang model with tunnel interface specifics, which is binding tunnel interface with a physical interface [RFC7223].



4. Design of GENERIC TUNNEL Modules

This document defines the YANG module "gen-tunnel", which augments the "interface" and "interface-state" lists defined in the "ietf-interfaces" module [RFC7223] with tunnel specific data nodes, and also adds tunnel specific state data.

4.1. tunnel configuration model

The data model has the following structure for tunnel configuration per interface:

```
module: gen-tunnel
      augment /if:interfaces/if:interface:
      +--rw tunnel-configuration
         +--rw base-interface?         if:interface-ref
         +--rw tunnel-name?            string
         +--rw ip-address?             yang:phys-address
         +--rw technology?             identityref
         +--rw encapsulation-method?   identityref
         +--rw (signaling-protocol-type)?
         |  +--:(signaling-protocol-null)
         |     +--rw signaling-null?         empty
         +--rw tunnel-source
         |  +--rw (tunnel-source)?
         |     +--:(interface)
         |     |  +--rw interface-type?   leafref
         |     |  +--rw interface?        if:interface-ref
         |     +--:(ipv4-address)
         |     |  +--rw ipv4-address?     inet:ipv4-address
         |     +--:(ipv6-address)
         |        +--rw ipv6-address?     inet:ipv6-address
         +--rw tunnel-destination
         |  +--rw (tunnel-destination)?
         |     +--:(interface)
         |     |  +--rw interface-type?   leafref
         |     |  +--rw interface?        if:interface-ref
         |     +--:(ipv4-address)
         |     |  +--rw ipv4-address?     inet:ipv4-address
         |     +--:(ipv6-address)
         |        +--rw ipv6-address?     inet:ipv6-address
         +--rw MTU?                    uint32
         +--rw ttl?                    uint8
         +--rw priority?               uint8
         +--rw (Multiplexing)?
         |  +--:(multiplexing-null)
         |     +--rw multiplexing-null?      empty
         +--rw (QoS)?
         |  +--:(Qos-null)
         |     +--rw QoS-null?               empty
         +--rw (Security)?
         |  +--:(Security-null)
         |     +--rw Security-null?          empty
         +--rw resource
               ......
```

o The base-interface which with type leafref is used pointing to the corresponding interface-name node in the interface yang [RFC7223]. As describe in section 3.1, each tunnel need to binding to a

physical interface[RFC7223]. In the gen-tunnel model, the base-interface leaf is playing this role.

- The tunnel-name is the identification of the tunnel. Different tunnel is distinguished via the leaf tunnel-name.

- The technology leaf indicate the technology of the tunnel such as IP, MPLS, VXLAN, etc. The default value is IP. And this allows easy extension of the YANG model by other technologies.

- The encapsulation-method leaf indicate the encapsulation method of the tunnel such as Minimal encapsulation, L2TP encapsulation, PPTP encapsulation, L2F encapsulation, UDP encapsulation, ATMP encapsulation, MSDP encapsulation, 6to4 encapsulation, 6over4 encapsulation, ISATAP encapsulation, etc. The default value is direct (no intermediate header). And this allows easy extension of the YANG model by other technologies.

- The signaling-protocol-type indicate the signaling protocol type of the tunnel such as rsvp, crldp, etc. And the default value is empty, which allows easy extension of the YANG model by other technologies.

- The multiplexing might be one which was explicitly designed for multiplexing, or one that wasn’t originally designed for this but can be pushed into service as a multiplexing field. For example: the Key field for GRE, the SPI field for IPsec, etc. In gen-tunnel model, the default value of multiplexing is empty, which allows easy extension of the yang model by other technologies.

- A tunnel may not have intrinsic QoS/SLA capabilities, but it inherits whatever mechanisms exist for IP. Other mechanisms such as RSVP extensions [RFC2764] or DiffServ extensions [RFC2983], may be used with it. In gen-tunnel model, the default value of QoS/ SLA capability is empty, which allows easy extension of the yang model by other technologies.

- A tunnel may provide significant security services. For example: When IPsec tunneling is used in conjunction with IPsec’s cryptographic capabilities, excellent authentication and integrity functions can be provided. In gen-tunnel model, the default value of security is empty, which allows easy extension of the yang model by other technologies.

4.1.1. Resource container

The resource container is used to indicate the resources required for a tunnel. Which contains a MaxRate leaf, a MaxBurstSize leaf, an ExBurstSize leaf, a Frequency leaf and a weight leaf.

```
module: gen-tunnel
augment /if:interfaces/if:interface:
   +--rw base-interface?         if:interface-ref
   ......
   +--rw resource
      +--rw bandwidth?      uint32
      +--rw MaxRate?        uint32
      +--rw MaxBurstSize?   uint32
      +--rw ExBurstSize?    uint32
      +--rw Frequency?      uint32
      +--rw weight?         uint32
```

- The bandwidth indicate the bandwidth required of this tunnel.
- The MaxRate indicate the maximum rate of this tunnel.
- The MaxBurstSize indicate the maximum burst size of this tunnel.
- The ExBurstSize indicate the excess burst size of this tunnel.
- The Frequency indicate the granularity of the availability of committed rate.
- The weight indicate the relative weight for using excess bandwidth above its committed rate.

4.2. tunnel state model

The tunnel state model augments the "interface-state" lists defined in the "ietf-interfaces" module [RFC7223] with tunnel specific state data. On the top of the tunnel state model, we defines a tunnel list, each tunnel within it corresponding to a tunnel instance. Within each tunnel, we present an admin-status, an oper-status , MTU, packets input/output, input/output error, input/output utility rate and the number of bytes.

```
module: gen-tunnel
augment /if:interfaces-state:
   +--ro tunnel-state
      +--ro tunnel* [tunnel-name]
         +--ro tunnel-name            leafref
         +--ro admin-status?          enumeration
         +--ro oper-status?           enumeration
         +--ro MTU?                   uint32
         +--ro packets-input?         yang:counter64
         +--ro input-errors?          yang:counter64
         +--ro packets-output?        yang:counter64
         +--ro output-errors?         yang:counter64
         +--ro input-utility-rate?    yang:counter64
         +--ro output-utility-rate?   yang:counter64
         +--ro transmitted-bytes?                yang:counter64
```

5. Gen-tunnel yang data hierarchy

The complete data hierarchy related to the Tunnel YANG model is presented below. The following notations are used within the data tree and carry the meaning as below.

Each node is printed as:

```
 <status> <flags> <name> <opts> <type>
```

```
        <status> is one of:
             +  for current
             x  for deprecated
             o  for obsolete


        <flags> is one of:

            rw for configuration data
            ro for non-configuration data
            -x for rpcs
            -n for notifications


        <name> is the name of the node
```

If the node is augmented into the tree from another module, its name is printed as <prefix>:<name>.

```
            <opts> is one of:

                 ?  for an optional leaf or choice
                 !  for a presence container
                 *  for a leaf-list or list
                 [<keys>] for a list's keys

            <type> is the name of the type for leafs and leaf-lists

   module: gen-tunnel
   augment /if:interfaces/if:interface:
            +--rw tunnel-configuration
               +--rw base-interface?         if:interface-ref
               +--rw tunnel-name?            string
               +--rw ip-address?             yang:phys-address
               +--rw technology?             identityref
               +--rw encapsulation-method?   identityref
               +--rw (signaling-protocol-type)?
               |  +--:(signaling-protocol-null)
               |     +--rw signaling-null?         empty
               +--rw tunnel-source
               |  +--rw (tunnel-source)?
               |     +--:(interface)
               |     |  +--rw interface-type?   leafref
               |     |  +--rw interface?        if:interface-ref
               |     +--:(ipv4-address)
               |     |  +--rw ipv4-address?     inet:ipv4-address
               |     +--:(ipv6-address)
               |        +--rw ipv6-address?     inet:ipv6-address
               +--rw tunnel-destination
               |  +--rw (tunnel-destination)?
               |     +--:(interface)
               |     |  +--rw interface-type?   leafref
               |     |  +--rw interface?        if:interface-ref
               |     +--:(ipv4-address)
               |     |  +--rw ipv4-address?     inet:ipv4-address
               |     +--:(ipv6-address)
               |        +--rw ipv6-address?     inet:ipv6-address
               +--rw MTU?                    uint32
               +--rw ttl?                    uint8
               +--rw priority?               uint8
               +--rw (Multiplexing)?
               |  +--:(multiplexing-null)
               |     +--rw multiplexing-null?      empty
               +--rw (QoS)?
               |  +--:(Qos-null)
               |     +--rw QoS-null?               empty
               +--rw (Security)?
```

```
               |  +--:(Security-null)
               |     +--rw Security-null?          empty
               +--rw resource
                  +--rw bandwidth?      uint32
                  +--rw MaxRate?        uint32
                  +--rw MaxBurstSize?   uint32
                  +--rw ExBurstSize?    uint32
                  +--rw Frequency?      uint32
                  +--rw weight?         uint32
         augment /if:interfaces-state:
            +--ro tunnel-state
               +--ro tunnel* [tunnel-name]
                  +--ro tunnel-name            leafref
                  +--ro admin-status?          enumeration
                  +--ro oper-status?           enumeration
                  +--ro MTU?                   uint32
                  +--ro packets-input?         yang:counter64
                  +--ro input-errors?          yang:counter64
                  +--ro packets-output?        yang:counter64
                  +--ro output-errors?         yang:counter64
                  +--ro input-utility-rate?    yang:counter64
                  +--ro output-utility-rate?   yang:counter64
                  +--ro transmitted-bytes?     yang:counter64
```

data hierarchy of TUNNEL

6. Tunnel YANG Module

```
   <CODE BEGINS> file "ietf-tunnel.yang"
      module gen-tunnel {
        namespace "http://example.com/gen-tunnel";
        prefix "tunnel";
        import ietf-yang-types { prefix yang;}
    import ietf-inet-types { prefix inet;}
        import ietf-interfaces {
          prefix if;
        }
        import iana-if-type {
          prefix ianaift;
        }
   organization "IETF Netmod Working Group";
        contact
          "wangzitao@huawei.com";
        description
          "This module defines ietf tunnel yang data model";

        revision 2015-06-09 {
```

```
        description
         "Initial revision. - 01 version";
         reference "draft-wwz-netmod-yang-tunnel-model-00";
        }
       identity technology-types {
        description
        "this is the base identity of technology types which are
         IP, GRE, MPLS, etc";
       }

       identity ipv4 {
        base technology-types;
        description
         "technology of ipv4";
       }

       identity ipv6 {
        base technology-types;
        description
         "technology of ipv6";
       }

   identity encapsulation-type{
    description
    "The encapsulation method used by a tunnel
    which are direct, GRE, 6to4, 6over4, IPsec, etc.";
   }

   identity direct{
    base encapsulation-type;
    description
    "no intermediate header.";
   }

   identity endpoint-type{
    description
    "this is the base identity of tunnel type which are CE, PE, etc.";
   }
        augment "/if:interfaces/if:interface" {
          when "if:type = 'ianaift:tunnel'";
     description
     "Parameters for configuring tunnel on interfaces.";
         container tunnel-configuration{
      description
      "tunnel configuration model.";
          leaf base-interface {
            type if:interface-ref;
            must "/if:interfaces/if:interface[if:name = current()]" {
```

```
              description
                "The base interface.";
            }
    description
                "The base interface.";
          }

      leaf tunnel-name{
      type string;
      description
      "this name can be used to distinguish different tunnel";
     }

      leaf ip-address{
       type yang:phys-address;
           description
   "ip-address of this tunnel interface";
      }
        leaf technology{
      type identityref{
       base technology-types;
      }
      description
      "this leaf can be used to indicate different technologies
       such as IP, GRE, MPLS, etc";
     }

         leaf encapsulation-method{
      type identityref{
       base encapsulation-type;
      }
      description
      "The encapsulation method used by a tunnel.";
     }


         leaf endpoint-type{
      type identityref{
       base endpoint-type;
      }
      description
      "The endpoint type.";
     }

       choice signaling-protocol-type {
         case signaling-protocol-null {
           description
             "this is a placeholder when no signaling protocol
```

```
     is needed";
           leaf signaling-null {
              type empty;
              description
                "there is no signaling protocol define,
     it will be defined in
     technology specific model.";
           }
         }
     description
     "signaling protocol type";
      }



       container tunnel-source{
    description
    "parameters of source tunnel";
        choice tunnel-source {
         case interface{
     leaf interface-type{
       type leafref{
    path "/if:interfaces/if:interface"
    +"/if:type";
   }
   description
   "interface type";
      }
      leaf interface{
      type if:interface-ref;
      description
       "Interface";
   }
     }
         case ipv4-address {
           leaf ipv4-address {
             type inet:ipv4-address;
             description
               "Ipv4 Address";
           }
           description
             "Ip Address based node Addressing.";
         }
         case ipv6-address {
           leaf ipv6-address {
             type inet:ipv6-address;
             description
               "Ipv6 Address";
```

```
          }
          description
            "ipv6 Address based node Addressing.";
        }
        description
          "node Addressing.";
      }
   }

   container tunnel-destination{
    description
    "Data nodes for the operational state of tunnel on interfaces";
        choice tunnel-destination {
         case interface{
     leaf interface-type{
      type leafref{
    path "/if:interfaces/if:interface"
    +"/if:type";
   }
   description
   "interface type";
      }
      leaf interface{
      type if:interface-ref;
      description
       "Interface";
   }
     }
         case ipv4-address {
           leaf ipv4-address {
             type inet:ipv4-address;
             description
               "Ipv4 Address";
           }
           description
             "Ip Address based node Addressing.";
         }
         case ipv6-address {
           leaf ipv6-address {
             type inet:ipv6-address;
             description
               "Ipv6 Address";
           }
           description
             "ipv6 Address based node Addressing.";
         }
         description
           "node Addressing.";
```

```
        }
       }

    leaf MTU{
    type uint32;
    description
    "Maximum Transmission Unit";
   }

   leaf ttl {
        type uint8;
        default "255";
        description
        "Time to Live";
       }

       leaf priority {
         type uint8;
         description
           "priority";
       }

   choice Multiplexing{
    description
    "multiplexing parameters";
    case multiplexing-null{
     description
         "this is a placeholder when no multiplexing is needed";
     leaf multiplexing-null{
      type empty;
      description
          "there is no multiplexing define, it will be defined
           in technology specific model.";
     }
    }
   }

      choice QoS{
    description
    "QoS Parameters";
    case Qos-null{
     description
         "this is a placeholder when no QoS is needed";
     leaf QoS-null{
      type empty;
      description
          "there is no QoS define, it will be defined
           in technology specific model.";
```

```
     }
    }
   }

   choice Security{
    description
    "security parameters";
    case Security-null{
     description
         "this is a placeholder when no Security is needed";
     leaf Security-null{
      type empty;
      description
          "there is no Security define, it will be defined
           in technology specific model.";
     }
    }
   }
   container resource{
    // if-feature resource-support;
     description
     "this container is used to indicate the resources required
          for a tunnel.";

      leaf bandwidth{
       type uint32;
   description
   "the bandwidth of tunnel";
      }

      leaf MaxRate{
       type uint32;
   description
   "The maximum rate in bits/second.";
      }

      leaf MaxBurstSize{
       type uint32;
   description
   "The maximum burst size in bytes.";
      }

      leaf ExBurstSize{
       type uint32;
   description
   "The Excess burst size in bytes.";
      }
```

```
      leaf Frequency{
       type uint32;
   description
   "The granularity of the availability of committed
            rate.";
      }

      leaf weight{
       type uint32;
   description
   "The relative weight for using excess bandwidth above
            its committed rate.";
      }
    }
        }
       }
        augment "/if:interfaces-state" {
          when "if:type = 'ianaift:tunnel'";
    description
    "Data nodes for the operational state of tunnel on interfaces";
    container tunnel-state{
   config false;
   description
   "define the state of this tunnel";
   list tunnel{
    key "tunnel-name";
    description
    "The list of tunnel on the interface";
    leaf tunnel-name{
     type leafref{
      path "/if:interfaces/if:interface"
     +"/tunnel:tunnel-configuration/tunnel:tunnel-name";
    }
     description
     "this leaf can be used to distinguish different tunnels";
    }

         leaf admin-status{
          type enumeration {
           enum up {
            value 1;
            description
            "Ready to pass packets.";
           }
           enum down {
            value 2;
            description
            "Not ready to pass packets and not in some test mode.";
```

```
            }
            enum testing {
             value 3;
             description
             "In some test mode.";
            }
           }
           description
           "The desired state of the tunnel.";
          }

          leaf oper-status {
           type enumeration {
            enum up {
             value 1;
             description
             "Ready to pass packets.";
            }
            enum down {
             value 2;
             description
             "The tunnel does not pass any packets.";
            }
            enum testing {
             value 3;
             description
             "In some test mode. No operational packets can
              be passed.";
            }
            enum unknown {
             value 4;
             description
             "Status cannot be determined for some reason.";
            }
            enum dormant {
             value 5;
             description
             "Waiting for some external event.";
            }
            enum not-present {
             value 6;
             description
             "Some component (typically hardware) is missing.";
            }
            enum lower-layer-down {
             value 7;
             description
             "Down due to state of lower-layer tunnel(s).";
```

```
            }
           }
           description
           "The current operational state of the tunnel.";
          }

        leaf MTU{
         type uint32;
      description
      "Maximum Transmit Uint";
        }

          leaf packets-input{
       type yang:counter64;
       description
       "Number of  packets input.";
      }

      leaf input-errors{
       type yang:counter64;
       description
       "Number of input packets dropped because of errors or for
            other reasons.";
      }

          leaf packets-output{
       type yang:counter64;
       description
       "Number of  packets output.";
      }

      leaf output-errors{
       type yang:counter64;
       description
       "Number of output packets dropped because of errors or for
            other reasons.";
      }

      leaf input-utility-rate{
       type yang:counter64;
       description
       "input utility rate.";
      }

      leaf output-utility-rate{
       type yang:counter64;
       description
       "output utility rate.";
```

```
     }

     leaf bytes{
      type yang:counter64;
      description
      "Number of bytes forwarded by the tunnel.";
     }
    }
   }
   }
      }
```

<CODE ENDS>

7. Security Considerations

The YANG module defined in this memo is designed to be accessed via the NETCONF protocol [RFC6241] . The lowest NETCONF layer is the secure transport layer and the mandatory-to-implement secure transport is SSH [RFC6242] . The NETCONF access control model [RFC6536] provides the means to restrict access for particular NETCONF users to a pre-configured subset of all available NETCONF protocol operations and content.

8. IANA Considerations

This document registers a URI in the IETF XML registry [RFC3688] . Following the format in RFC 3688, the following registration is requested to be made:

URI: urn:ietf:params:xml:ns:yang:ietf-tunnel

Registrant Contact: The IESG.

XML: N/A, the requested URI is an XML namespace.

This document registers a YANG module in the YANG Module Names registry [RFC6020].

name: ietf-tunnel

namespace:urn:ietf:params:xml:ns:yang:ietf-tunnel

prefix: itun reference: RFC XXXX

9. Normative References

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

[RFC6020] Bjorklund, M., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, October 2010.

Authors' Addresses

Aijun Wang
China Telecom
No.118,Xizhimenneidajie,Xicheng District
Beijing 100035
China

Email: wangaj@ctbri.com.cn

Zitao Wang
Huawei Technologies,Co.,Ltd
101 Software Avenue, Yuhua District
Nanjing 210012
China

Email: wangzitao@huawei.com

Yan Zhuang
Huawei
101 Software Avenue, Yuhua District
Nanjing, Jiangsu 210012
China

Email: zhuangyan.zhuang@huawei.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: July 24, 2016

L. Zheng, Ed.
Huawei Technologies
C. Pignataro
R. Penno
Cisco Systems, Inc.
Z. Wang
Huawei Technologies

January 21, 2016

Yang Data Model for Generic Routing Encapsulation (GRE)
draft-zheng-intarea-gre-yang-01.txt

Abstract

This document defines a YANG data model that can be used to configure and manage Generic Routing Encapsulation (GRE).

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 July 24, 2016.

Copyright Notice

Table of Contents

1. Introduction

Generic Routing Encapsulation (GRE) [RFC2784] specifies a protocol for encapsulation of an arbitrary network layer protocol over another arbitrary network layer protocol. YANG [RFC6020] is a data definition language that was introduced to define the contents of a conceptual data store that allows networked devices to be managed using NETCONF [RFC6241]. This document defines a YANG data model that can be used to configure and manage GRE.

The rest of this document is organized as follows. Section 2 presents the scope of this document. Section 3 provides the design of the GRE configuration data model in details. Section 4 presents the complete data hierarchy of GRE YANG model. Section 5 specifies the YANG module and section 6 lists examples which conform to the YANG module specified in this document. Finally, security considerations are discussed in Section 7.

2. Scope

The fundemantel protocol of GRE is defined in [RFC2784]. [RFC2890] describes extensions by which two fields, Key and Sequence Number, can be optionally carried in the GRE Header.
[I-D.ietf-intarea-gre-ipv6] specifies GRE procedures for IPv6, used

as either the payload or delivery protocol.
[I-D.ietf-intarea-gre-mtu] describes how vendors have solved the GRE fragmentation problem. These RFCs and documents are considered in this Yang Module.

3. Design of the Data Model

This YANG data model is defined to be used to configure and manage Generic Routing Encapsulation (GRE) . Under the top level container is the list gre-tunnel, the leaf tunnel-name is used as the key for the list. Under the list, nodes are defined to enable the tunnel encapsulation configuration when either IPv4 or IPv6 is used as the delivery protocol. Nodes are also defined to enable the checksum bit set, tunnel fragmentation, Path MTU Discovery, Key and Key value set, and Sequence Number configuration respectively, based on various GRE RFCs and documents which are summarized in Section 2.

4. Data Hierarchy

The complete data hierarchy of GRE YANG model is presented below.

```
module: ietf-gre
   +--rw gre-tunnel
      +--rw gre-tunnel* [tunnel-name]
         +--rw tunnel-name               string
         +--rw (delivery-protocol)?
         |  +--:(ipv4)
         |  |  +--rw source-ipv4-address?      inet:ipv4-address
         |  |  +--rw dest-ipv4-address?        inet:ipv4-address
         |  +--:(ipv6)
         |     +--rw source-ipv6-address?      inet:ipv6-address
         |     +--rw dest-ipv6-address?        inet:ipv6-address
         +--rw pmtud-enable?             boolean
         +--rw fragmentation-enable?     boolean
         +--rw checksum-enable?          boolean
         +--rw key-enable?               boolean
         +--rw key?                      uint32
         +--rw sequence-number-enable?   boolean
```

5. GRE Yang Module

```
<CODE BEGINS> file "ietf-gre@2015-07-02.yang"
module ietf-gre {
  namespace "urn:ietf:params:xml:ns:yang:ietf-gre";
  //namespace to be assigned by IANA
  prefix "gre";
  import ietf-inet-types {
```

```
      prefix "inet";
    }
    organization "IETF INTAREA Working Group";
    contact "draft-zheng-intarea-gre-yang";
    description "This module contains the YANG definition for GRE
                 parameters as per RFC2784, RFC2890,
                 draft-ietf-intarea-gre-ipv6 and
                 draft-ietf-intarea-gre-mtu";
    revision "2015-07-02" {
     description "Initial revision.";
     reference "draft-zheng-intarea-gre-yang";
    }

    container gre-tunnel {
      description "Top level container";
      list gre-tunnel {
        key "tunnel-name";
        description "GRE tunnel";
        leaf tunnel-name {
          type string {
            length "1..63";
          }
          description "GRE tunnel name";
        }
        choice delivery-protocol {
          case ipv4 {
            leaf source-ipv4-address {
              type inet:ipv4-address;
              description "Source IP address";
            }
            leaf dest-ipv4-address {
              type inet:ipv4-address;
              description "Destination IP address";
            }
          }
          case ipv6 {
            leaf source-ipv6-address {
              type inet:ipv6-address;
              description "Source IP address";
            }
            leaf dest-ipv6-address {
              type inet:ipv6-address;
              description "Destination IP address";
            }
          }
          description "Delivery protocol";
        }
        leaf pmtud-enable {
```

```
            type boolean;
            description "Enable tunnel PMTU discovery";
          }
          leaf fragmentation-enable {
            type boolean;
            description "Enable delivery packets fragmentation";
          }
          leaf checksum-enable {
            type boolean;
            description "Enable GRE tunnel checksum verification";
          }
          leaf key-enable {
            type boolean;
            description "Enable optional GRE tunnel Key";
          }
          leaf key {
            when "/gre-tunnel/gre-tunnel/key-enable == 'true'" {
              description "When key-enable is true";
            }
            type uint32;
            description "GRE tunnel key value";
          }
          leaf sequence-number-enable {
            type boolean;
            description "Enable optional GRE tunnel Sequence Number";
          }
        }
      }
    }
    <CODE ENDS>
```

6. Examples

Examples of using Yang module to configure and manage GRE will be given here in the update when the Yang module is stable.

7. Security Considerations

The configuration and state data defined in this document is designed to be accessed via the NETCONF protocol [RFC6241]. The lowest NETCONF layer is the secure transport layer and the mandatory-to-implement secure transport is SSH [RFC6242]. The authors recommend to implement the NETCONF access control model [RFC6536] to restrict access for particular NETCONF users to a pre-configured subset of all available NETCONF protocol operations and content.

There are a number of config true nodes defined in the YANG module which are writable/creatable/deletable. These data nodes may be

considered sensitive or vulnerable in some network environments. Write operations to these data nodes without proper protection can have a negative effect on network operations.

8. IANA Considerations

The IANA is requested to assign a new namespace URI from the IETF XML registry.

URI:TBA

9. Acknowledgements

We would also like to thank XXX.

10. References

10.1. Normative References

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

[RFC2784] Farinacci, D., Li, T., Hanks, S., Meyer, D., and P. Traina, "Generic Routing Encapsulation (GRE)", RFC 2784, DOI 10.17487/RFC2784, March 2000, <http://www.rfc-editor.org/info/rfc2784>.

[RFC2890] Dommety, G., "Key and Sequence Number Extensions to GRE", RFC 2890, DOI 10.17487/RFC2890, September 2000, <http://www.rfc-editor.org/info/rfc2890>.

[RFC6020] Bjorklund, M., Ed., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, DOI 10.17487/RFC6020, October 2010, <http://www.rfc-editor.org/info/rfc6020>.

10.2. Informative References

[I-D.ietf-intarea-gre-ipv6] Pignataro, C., Bonica, R., and S. Krishnan, "IPv6 Support for Generic Routing Encapsulation (GRE)", draft-ietf-intarea-gre-ipv6-14 (work in progress), September 2015.

[I-D.ietf-intarea-gre-mtu]
Bonica, R., Pignataro, C., and J. Touch, "A Widely-Deployed Solution To The Generic Routing Encapsulation (GRE) Fragmentation Problem", draft-ietf-intarea-gre-mtu-05 (work in progress), May 2015.

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

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

[RFC6536] Bierman, A. and M. Bjorklund, "Network Configuration Protocol (NETCONF) Access Control Model", RFC 6536, DOI 10.17487/RFC6536, March 2012, <http://www.rfc-editor.org/info/rfc6536>.

Authors' Addresses

Lianshu Zheng (editor)
Huawei Technologies
China

Email: vero.zheng@huawei.com

Carlos Pignataro
Cisco Systems, Inc.
USA

Email: cpignata@cisco.com

Reinaldo Penno
Cisco Systems, Inc.
USA

Email: repenno@cisco.com

Zishun Wang
Huawei Technologies
China

Email: wangzishun@huawei.com