

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.

Copyright Notice

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

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

Table of Contents

1. Introduction	2
1.1. Requirements Notation	3
2. Problem Statement	3
2.1. Graceful Key Rollover using Key Chains	3
3. Design of the Key Chain Model	4
3.1. Key Chain Operational State	5
3.2. Key Chain Model Features	5
3.3. Key Chain Model Tree	5
4. Key Chain YANG Model	8
5. Relationship to other Work	16
6. Security Considerations	16
7. IANA Considerations	16
8. References	17
8.1. Normative References	17
8.2. Informative References	17
Appendix A. Acknowledgments	18
Authors' Addresses	18

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 keystack {
      leaf keystack {
        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

Routing Area Working Group
Internet-Draft
Intended status: Informational
Expires: January 4, 2016

C. Bowers
Juniper Networks
J. Farkas
Ericsson
July 3, 2015

Applicability of Maximally Redundant Trees to IEEE 802.1Qca Path Control
and Reservation
draft-bowers-rtgwg-mrt-applicability-to-8021qca-01

Abstract

IEEE 802.1Qca Path Control and Reservation (PCR) [IEEE8021Qca] uses the algorithm specified in [I-D.ietf-rtgwg-mrt-frr-algorithm] to compute Maximally Redundant Trees (MRTs) to be used for the protection of data traffic in bridged networks. This document discusses the applicability of the MRT algorithm to 802.1Qca.

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.

Copyright Notice

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

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of

the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
2. How 802.1Qca uses the MRT algorithm	3
2.1. MRT Explicit Trees in 802.1Qca	3
2.2. 'MRT with GADAG' Explicit Trees in 802.1Qca	3
2.3. MRT Explicit Trees as Strict Trees	4
3. Other considerations	4
3.1. Unequal link metrics	4
3.2. Computation of MRT-Blue and MRT-Red next-hops from the point of view of other nodes	5
3.3. Recalculation of MRTs	5
4. IANA Considerations	6
5. Security Considerations	6
6. Acknowledgements	6
7. Informative References	6
Authors' Addresses	7

1. Introduction

IEEE 802.1Qca Shortest Path Bridging (SPB) [IEEE8021ca] is an amendment to IEEE Std 802.1Q that allows bridged frames to travel on the shortest path between their source and destination(s), as opposed to traveling along paths determined by shared spanning trees. [IEEE8021ca] and [RFC6329] specify extensions to IS-IS that allow bridges to share the topology information needed to construct shortest path trees. These extensions are referred to here as ISIS-SPB. [IEEE8021ca] has been already incorporated in [IEEE8021Q].

[IEEE8021Qca] is an amendment to [IEEE8021Q] that specifies explicit path control, bandwidth assignment, and protection mechanisms for data flows for bridged networks. [IEEE8021Qca] is an extension to IS-IS that builds upon the ISIS-SPB extensions and extends them further as described in [I-D.ietf-isis-pcr]. These extensions (referred to here as ISIS-PCR) allow bridges to share the information needed to construct explicit trees.

[IEEE8021Qca] specifies five different methods for the construction of explicit trees as well as how to share the information needed to construct these trees. These five different methods of explicit trees are referred to as Strict Tree, Loose Tree, Loose Tree Set, MRT, and MRT with GADAG. This document is concerned with the MRT and 'MRT with GADAG' explicit tree methods (algorithms). Both methods produce Maximally Redundant Trees (MRTs) [I-D.ietf-rtgwg-mrt-frr-architecture].

This document is intended to explain the relationship between [IEEE8021Qca] and [I-D.ietf-rtgwg-mrt-frr-algorithm]. The text should not be interpreted as normative with respect to either.

2. How 802.1Qca uses the MRT algorithm

The algorithm for computing Maximally Redundant Trees in [I-D.ietf-rtgwg-mrt-frr-algorithm] has been specified with a focus on supporting fast-reroute for the protection of unicast IP and LDP traffic, as described in [I-D.ietf-rtgwg-mrt-frr-architecture]. The computation described in [I-D.ietf-rtgwg-mrt-frr-algorithm] starts with the topology of an IGP area, prunes it to form an MRT Island topology, constructs a GADAG, and then uses that GADAG to construct a complete set of destination-based MRT-Blue and MRT-Red trees rooted at each node in the MRT Island, where each tree spans all nodes in the MRT Island. [IEEE8021Qca] supports this mode of operation, but it also supports other modes of operation as described below.

2.1. MRT Explicit Trees in 802.1Qca

In [IEEE8021Qca], the flooding of an SPB Instance sub-TLV ([RFC6329]) with the MRT ECT Algorithm value in the absence of a Topology sub-TLV ([I-D.ietf-isis-pcr]) results in the creation of an MRT-Blue and an MRT-Red tree rooted at each node in the domain, and each tree spans all nodes in the domain. This is equivalent to the behavior described in [I-D.ietf-rtgwg-mrt-frr-algorithm].

In addition, [IEEE8021Qca] allows one to affect both the number and structure of the MRT-Blue and Red trees by including the Topology sub-TLV in the MT-Capability TLV (type 144 [RFC6329]). In the context of the MRT ECT Algorithm, Hop sub-TLVs in the Topology sub-TLV specify nodes with flags that can indicate Root, Exclude, or Edge Bridge. In the context of the MRT algorithm, all nodes with the Exclude flag set are excluded from the MRT Island. The GADAG is then computed for the MRT Island. For each node with the Root flag set, an MRT-Blue and an MRT-Red tree rooted at that node is constructed.

Note that the Edge Bridge flag does not affect the construction of the MRTs. It is used to determine which filtering database (FDB) entries to install once the trees have been determined.

2.2. 'MRT with GADAG' Explicit Trees in 802.1Qca

In the previous section, each node will compute the same GADAG based on the same topology information using the algorithm steps in sections 5.4 and 5.5 of [I-D.ietf-rtgwg-mrt-frr-algorithm]. Each node then applies the algorithm steps in section 5.6.5 of

[I-D.ietf-rtgwg-mrt-frr-algorithm] to that GADAG, in order to compute the MRT-Blue and MRT-Red next-hops for the trees of interest.

[IEEE8021Qca] can operate in a mode where each node is supplied with a common GADAG (communicated via ISIS-PCR), from which each node then determines the MRT-Blue and MRT-Red next-hops for the trees of interest. That is, this mode of operation bypasses the algorithm steps in section 5.4 and 5.5 of [I-D.ietf-rtgwg-mrt-frr-algorithm] and only applies the algorithm steps in section 5.6.5 to the GADAG communicated directly via ISIS-PCR.

This behavior is controlled in [IEEE8021Qca] by flooding an SPB Instance sub-TLV with the 'MRT with GADAG' ECT Algorithm value as well as a Topology sub-TLV. Hop sub-TLVs in this Topology sub-TLV are used to describe the common GADAG using an ear decomposition. The first Hop sub-TLV is the GADAG root followed by a sequence of Hop sub-TLVs describing an ordered ear that terminates on the GADAG root. Subsequent ears are described as sequences of Hop sub-TLVs. Setting the Root flag for a given Hop sub-TLV indicates that MRT-Blue and Red trees rooted at that node should be constructed.

2.3. MRT Explicit Trees as Strict Trees

A Path Computation Element can also implement each computation step of [I-D.ietf-rtgwg-mrt-frr-algorithm] and compute the MRTs. The MRTs can be then specified by Topology sub-TLVs, one for each. The SPB Instance sub-TLV then conveys the ST ECT Algorithm value.

3. Other considerations

3.1. Unequal link metrics

[IEEE8021aq] specifies that if two SPB Link Metrics are different at each end of a link, the maximum of the two values is used in SPB calculations. In order to provide symmetry and maintain consistency with [IEEE8021aq], [IEEE8021Qca] places the same requirement on the Link Metrics for the topology graph that is used in the MRT algorithm. In order to accomplish this, [IEEE8021Qca] makes the same modification to link metrics before applying the MRT algorithm. This change of metric values can affect the ordering of interfaces on a given node through the Interface_Compare function in section 5 of [I-D.ietf-rtgwg-mrt-frr-algorithm], which in turn can affect the GADAG computed. The change of metric values can also affect the results of the SPF_No_Traverse_Root function used in determining MRT next-hops, since the SPF traversal depends on metric values.

This modification of link metrics applies ONLY to [IEEE8021Qca]. When the MRT lowpoint inheritance algorithm in

[I-D.ietf-rtgwg-mrt-frr-algorithm] is applied to IP/LDP FRR, the topology graph with the link metrics advertised by the IGP are used without modification by the MRT algorithm.

3.2. Computation of MRT-Blue and MRT-Red next-hops from the point of view of other nodes

In the algorithm described in [I-D.ietf-rtgwg-mrt-frr-algorithm] a given node computes and installs its own MRT-Blue and MRT-Red next-hops for all destinations. This computation is all that is required for the IP/LDP FRR application to function properly. An individual node does not need to compute the MRT-Blue and MRT-Red next-hops used by other nodes. Instead the complete MRT tree structure is created in the network as the result of each node computing and installing the appropriate MRT next-hops. This is analogous to the way that shortest path trees are instantiated in shortest path routing, with each node needing to compute and install only its own shortest path next-hops.

In some scenarios, a bridge using [IEEE8021Qca] may need to know more than just its own MRT-Blue and MRT-Red next-hops. This can be accomplished by having a bridge perform the MRT next-hop computation specified in section 5.6.5 of [I-D.ietf-rtgwg-mrt-frr-algorithm] from the point of view of one or more other bridges. The result of computing an MRT next-hop from the point of view of another bridge is the normative result. An implementation may use another method to compute MRT next-hops from the point of view of remote bridges as long as it produces the same result.

This does not modify the MRT algorithm with respect to its use for the IP/LDP FRR application as described in [I-D.ietf-rtgwg-mrt-frr-architecture].

3.3. Recalculation of MRTs

MRTs can be used for the protection of SPTs in a bridged network similarly to IP/LDP FRR application of MRTs. A pair of MRT-Blue and MRT-Red then protect the SPT rooted at the MRT Root. The MRTs are only used for protection, i.e. MRTs do not carry traffic during normal operation, similarly to IP/LDP FRR operations. The Point of Local Repair (PLR) is responsible for redirecting traffic from SPTs to MRTs upon detection of a failure event.

In 802.1Qca, recalculation of MRTs after a topology change follows the general method specified in Section 12.2 of [I-D.ietf-rtgwg-mrt-frr-architecture], with an additional requirement. Immediately after a failure, the PLR or PLRs redirect some traffic onto MRTs. In the meantime, all nodes receive

notification of the failure, recompute SPTs, and install them in their FIBs. The PLRs take the traffic off of the MRTs, and put the traffic on the new SPTs. Based on [I-D.ietf-rtgwg-mrt-frr-architecture], at this point it is safe to recompute and install the new MRTs corresponding to the new topology.

802.1Qca places an additional requirement on when it is safe to install the new MRTs. The new MRTs should not be recomputed and installed if there is any reason to suspect that the nodes of the domain do not share a common view on the network topology. This is in order to prevent loops in the MRT paths that may be used by PLRs at the next failure event. The loop prevention method to be used for MRTs in 802.1Qca is the Agreement Protocol, which is specified in [IEEE8021aq] and also described in [AP].

Note that while 802.1Qca requires that the Agreement Protocol be used to avoid loops on MRTs, it does not mandate the use of the Agreement Protocol for the shortest path trees, where other loop mitigation techniques can be used.

4. IANA Considerations

This document introduces no new IANA Considerations.

5. Security Considerations

The ISIS-PCR extensions for the use of the MRT algorithm are not believed to introduce new security concerns.

6. Acknowledgements

The authors would like to thank Alvaro Retana for his suggestions and review.

7. Informative References

[AP] Seaman, M., "Agreement Protocol", September 7, 2010, <<http://www.ieee802.org/1/files/public/docs2010/aq-seaman-agreement-protocol-0910-v2.pdf>>.

[I-D.ietf-isis-pcr] Farkas, J., Bragg, N., Unbehagen, P., Parsons, G., Ashwood-Smith, P., and C. Bowers, "IS-IS Path Computation and Reservation", draft-ietf-isis-pcr-00 (work in progress), April 2015.

[I-D.ietf-rtgwg-mrt-frr-algorithm]

Envedi, G., Csaszar, A., Atlas, A., Bowers, C., and A. Gopalan, "Algorithms for computing Maximally Redundant Trees for IP/LDP Fast- Reroute", draft-ietf-rtgwg-mrt-frr-algorithm-02 (work in progress), January 2015.

[I-D.ietf-rtgwg-mrt-frr-architecture]

Atlas, A., Kebler, R., Bowers, C., Envedi, G., Csaszar, A., Tantsura, J., and R. White, "An Architecture for IP/LDP Fast-Reroute Using Maximally Redundant Trees", draft-ietf-rtgwg-mrt-frr-architecture-05 (work in progress), January 2015.

[IEEE8021aq]

IEEE 802.1, "IEEE 802.1aq: IEEE Standard for Local and metropolitan area networks - Media Access Control (MAC) Bridges and Virtual Bridged Local Area Networks - Amendment 20: Shortest Path Bridging", 2012, <<http://standards.ieee.org/getieee802/download/802.1aq-2012.pdf>>.

[IEEE8021Q]

IEEE 802.1, "IEEE 802.1Q-2014: IEEE Standard for Local and metropolitan area networks - Bridges and Bridged Networks", 2014, <<http://standards.ieee.org/findstds/standard/802.1Q-2014.html>>.

[IEEE8021Qca]

IEEE 802.1, "IEEE 802.1Qca Bridges and Bridged Networks - Amendment: Path Control and Reservation - Draft 2.1", (work in progress), June 23, 2015, <<http://www.ieee802.org/1/pages/802.1ca.html>>.

[RFC6329] Fedyk, D., Ashwood-Smith, P., Allan, D., Bragg, A., and P. Unbehagen, "IS-IS Extensions Supporting IEEE 802.1aq Shortest Path Bridging", RFC 6329, April 2012.

Authors' Addresses

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

Email: cbowers@juniper.net

Janos Farkas
Ericsson
Konyves Kalman krt. 11/B
Budapest 1097
Hungary

Email: janos.farkas@ericsson.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 10, 2015

B. Decraene
Orange
March 9, 2015

Back-off SPF algorithm for link state IGP
draft-decraene-rtgwg-backoff-algo-01

Abstract

This document defines a standard algorithm to back-off link-state IGP SPF computations.

Having one standardized algorithm improves interoperability by reducing the probability and/or duration of transient forwarding loops during the IGP convergence in the area/level when the network reacts to multiple consecutive events.

Requirements Language

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

Status of This Memo

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

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

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

This Internet-Draft will expire on September 10, 2015.

Copyright Notice

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

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

1. Introduction

Link state IGP, such as IS-IS [ISO10589-Second-Edition] and OSPF [RFC2328], performs distributed computation on all nodes of the area/level. In order to have consistent routing tables across the network, such distributed computation requires that all routers have the same vision of the network (Link State DataBase (LSDB)) and perform their computation at the same time.

In general, when the network is stable, there is a desire to compute the new SPF as soon as the failure is known, in order to quickly route around the failure. However, when the network is experiencing multiple consecutive failures over a short period of time, there is a desire to limit the frequency of SPF computations. Indeed, this allow reducing the control plane resources used by IGP and all protocols/sub system reacting on it such as LDP, RSVP-TE, BGP, Fast ReRoute computations, FIB updates..., reduce the churn on nodes and in the network, in particular reduce side effects such as micro-loops which may happen during each IGP convergence.

To allow for this, some back-off algorithm have been implemented. Different implementations choose different algorithms, hence in a multi-vendor network, it's not possible to enforce that all routers triggers their SPF computation after the same waiting delay. This situation increases the average differential delay between routers end of RIB computation. It also increases the probability that different routers compute their RIB based on a different LSDB. Both increases the probability and/or duration of micro-loops.

To allow for multi-vendors networks having all the routers delaying their SPF for the same duration, this document specifies a standardized algorithm. Implementations may offer alternative optional algorithms.

2. High level goals

The high level goals of this algorithm are the following:

- o Very fast convergence for single simple events (link failure).
- o Fast convergence in general while the IGP stability is considered under control.
- o A long delay when the IGP stability is considered out of control, in order to let all related process calm down.
- o At any time, try to avoid using different SPF_TIMERS values for nodes in the area/level. Even though not all nodes will receive IGP message at the same time (due to difference in distance from the source and due to different flooding implementations on the path from the source).

3. Definitions and parameters

IGP events: An LSDB change requiring a new RIB computation (topology change, prefix change, metric change). No distinction is done between the type of computation performed (e.g. full SPF, incremental SPF, PRC). The type of computation is a local consideration.

The SPF_DELAY timer can take the following values:

INITIAL_WAIT: a very small delay to quickly handle link failure. e.g. 0 millisecond.

FAST_WAIT: a small delay to have a fast convergence. e.g. 50-100 millisecond. Note: we want to be fast, but as this failure requires multiple IGP events, being too fast increase the probability to receive additional IGP events just after the RIB computation.

LONG_WAIT: a long delay as IGP is unstable. e.g. 2 seconds. Note: let's bring calm in the IGP.

The TIME_TO_CONVERGE timer is the time to learn all the IGP events related to a single failure (e.g. node failure, SRLG failure). e.g. 1 second. It's mostly dependent on variation of failure detection times between all nodes which are neighbour to the failure, and then may depend on different flooding algorithms of nodes in the network.

The HOLD_DOWN timer is the time needed with no IGP events received, before considering that the IGP is quiet again and we can set the SPF_DELAY back to INITIAL_WAIT. e.g. 5 seconds.

4. Principle of SPF delay algorithm

The first IGP event is handled very quickly (`INITIAL_WAIT`) in order to be very reactive for the first event if it only needs one IGP event (e.g. link failure, prefix change).

If more IGP events are received quickly after, we consider that they are related to the same single failure, and handle the IGP events relatively quickly (`FAST_WAIT`) during the time needed to receive all the IGP events related to the failure (`TIME_TO_CONVERGE`).

If IGP events are still received after this time, then the network is presumably experiencing multiple independent failures and the while waiting for its stability, the computations are delayed for a longer time (`LONG_WAIT`).

Note: previous SPF delay algorithms used to count the number of RIB computations. However, as all nodes may receive the LSP events in a different way we cannot assume that all nodes will perform the same number of SPF computations or that they will schedule them at the same time. For example, assuming that the SPF delay is 50 ms, node R1 may receive 3 IGP events (E1, E2, E3) in those 50 ms and hence will perform a single routing computation. While another node R2 may only receive 2 events (E1, E2) in those 50ms and hence will schedule another routing computation when further receiving E3. That's why this document prefers to define a time limit (`TIME_TO_CONVERGE`) since the first event, rather than a number of routing computations.

5. Specification of SPF delay algorithm

When the previous IGP events is more than `HOLD_DOWN` ago:

- o The IGP is set to the `QUIET` state.

When the IGP is in the `QUIET` state and an IGP event is received:

- o The time of this first IGP event is stored in `FIRST_EVENT_TIME`.
- o The next RIB computation time is set to LSP receive time + `INITIAL_WAIT`.
- o The IGP is set to the `FAST_WAIT` state.

When the IGP is in the `FAST_WAIT` state and an IGP event is received:

- o If more than `TIME_TO_CONVERGE` has passed since `FIRST_EVENT_TIME`, then the IGP is set to the `HOLD_DOWN` state.

- o If the next RIB_computation time is in the past, set the next RIB computation time to LSP receive time + FAST_WAIT.

When the IGP is in the HOLD_DOWN state and an IGP event is received:

- o If the next RIB_computation time is in the past, set the next RIB computation time to LSP receive time + LONG_WAIT.

6. Impact on micro-loops

Micro-loops during IGP convergence are due to a non synchronized or non ordered update of the forwarding information tables (FIB) [RFC5715] [RFC6976] [I-D.litkowski-rtgwg-spf-uloop-pb-statement]. FIB are installed after multiple steps such as SPF wait time, SPF computation, FIB distribution and FIB update. This document only address the first contribution. This standardized procedure reduces the probability and/or duration of micro-loops when the IGP experience multiple consecutive events. It does not remove all micro-loops. However, it is beneficial and its cost seems limited compared to full solutions such as [RFC5715] or [RFC6976].

7. IANA Considerations

No IANA actions required.

8. Security considerations

This document has no impact on the security of the IGP.

9. Acknowledgements

We would like to acknowledge Hannes Gredler, Les Ginsberg and Pierre Francois for the discussions related to this document.

10. References

10.1. Normative References

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

10.2. Informative References

[I-D.litkowski-rtgwg-spf-uloop-pb-statement]
Litkowski, S., "Link State protocols SPF trigger and delay algorithm impact on IGP microloops", draft-litkowski-rtgwg-spf-uloop-pb-statement-02 (work in progress), March 2015.

[ISO10589-Second-Edition]

International Organization for Standardization,
"Intermediate system to Intermediate system intra-domain
routing 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.

[RFC2328] Moy, J., "OSPF Version 2", STD 54, RFC 2328, April 1998.

[RFC5715] Shand, M. and S. Bryant, "A Framework for Loop-Free
Convergence", RFC 5715, January 2010.

[RFC6976] Shand, M., Bryant, S., Previdi, S., Filsfils, C.,
Francois, P., and O. Bonaventure, "Framework for Loop-Free
Convergence Using the Ordered Forwarding Information Base
(oFIB) Approach", RFC 6976, July 2013.

Author's Address

Bruno Decraene
Orange
38 rue du General Leclerc
Issy Moulineaux cedex 9 92794
France

Email: bruno.decraene@orange.com

NETMOD Working Group
Internet-Draft
Intended status: Standards Track
Expires: May 7, 2017

L. Lhotka
CZ.NIC
A. Lindem
Cisco Systems
November 03, 2016

A YANG Data Model for Routing Management
draft-ietf-netmod-routing-cfg-25

Abstract

This document contains a specification of three YANG modules and one submodule. Together they form the core routing data model which serves as a framework for configuring and managing a routing subsystem. It is expected that these modules will be augmented by additional YANG modules defining data models for control plane protocols, route filters and other functions. The core routing data model provides common building blocks for such extensions -- routes, routing information bases (RIB), and controlplane protocols.

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, 2017.

Copyright Notice

Copyright (c) 2016 IETF Trust and the persons identified as the document authors. All rights reserved.

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

to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	3
2. Terminology and Notation	4
2.1. Glossary of New Terms	5
2.2. Tree Diagrams	5
2.3. Prefixes in Data Node Names	6
3. Objectives	6
4. The Design of the Core Routing Data Model	7
4.1. System-Controlled and User-Controlled List Entries	8
5. Basic Building Blocks	9
5.1. Route	9
5.2. Routing Information Base (RIB)	9
5.3. Control Plane Protocol	10
5.3.1. Routing Pseudo-Protocols	10
5.3.2. Defining New Control Plane Protocols	11
5.4. Parameters of IPv6 Router Advertisements	12
6. Interactions with Other YANG Modules	13
6.1. Module "ietf-interfaces"	13
6.2. Module "ietf-ip"	13
7. Routing Management YANG Module	14
8. IPv4 Unicast Routing Management YANG Module	26
9. IPv6 Unicast Routing Management YANG Module	32
9.1. IPv6 Router Advertisements Submodule	37
10. IANA Considerations	47
11. Security Considerations	49
12. Acknowledgments	49
13. References	49
13.1. Normative References	50
13.2. Informative References	50
Appendix A. The Complete Data Trees	51
A.1. Configuration Data	51
A.2. State Data	53
Appendix B. Minimum Implementation	54
Appendix C. Example: Adding a New Control Plane Protocol	54
Appendix D. Data Tree Example	57
Appendix E. Change Log	65
E.1. Changes Between Versions -24 and -25	65
E.2. Changes Between Versions -23 and -24	65
E.3. Changes Between Versions -22 and -23	65
E.4. Changes Between Versions -21 and -22	66
E.5. Changes Between Versions -20 and -21	66
E.6. Changes Between Versions -19 and -20	66

E.7.	Changes Between Versions -18 and -19	66
E.8.	Changes Between Versions -17 and -18	66
E.9.	Changes Between Versions -16 and -17	67
E.10.	Changes Between Versions -15 and -16	67
E.11.	Changes Between Versions -14 and -15	68
E.12.	Changes Between Versions -13 and -14	68
E.13.	Changes Between Versions -12 and -13	68
E.14.	Changes Between Versions -11 and -12	69
E.15.	Changes Between Versions -10 and -11	69
E.16.	Changes Between Versions -09 and -10	70
E.17.	Changes Between Versions -08 and -09	70
E.18.	Changes Between Versions -07 and -08	70
E.19.	Changes Between Versions -06 and -07	70
E.20.	Changes Between Versions -05 and -06	71
E.21.	Changes Between Versions -04 and -05	71
E.22.	Changes Between Versions -03 and -04	72
E.23.	Changes Between Versions -02 and -03	72
E.24.	Changes Between Versions -01 and -02	73
E.25.	Changes Between Versions -00 and -01	73
Authors' Addresses		74

1. Introduction

This document contains a specification of the following YANG modules:

- o Module "ietf-routing" provides generic components of a routing data model.
- o Module "ietf-ipv4-unicast-routing" augments the "ietf-routing" module with additional data specific to IPv4 unicast.
- o Module "ietf-ipv6-unicast-routing" augments the "ietf-routing" module with additional data specific to IPv6 unicast. Its submodule "ietf-ipv6-router-advertisements" also augments the "ietf-interfaces" [RFC7223] and "ietf-ip" [RFC7277] modules with IPv6 router configuration variables required by [RFC4861].

These modules together define the so-called core routing data model, which is intended as a basis for future data model development covering more sophisticated routing systems. While these three modules can be directly used for simple IP devices with static routing (see Appendix B), their main purpose is to provide essential building blocks for more complicated data models involving multiple control plane protocols, multicast routing, additional address families, and advanced functions such as route filtering or policy routing. To this end, it is expected that the core routing data model will be augmented by numerous modules developed by other IETF working groups.

2. Terminology and 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 [RFC2119].

The following terms are defined in [RFC6241]:

- o client,
- o message,
- o protocol operation,
- o server.

The following terms are defined in [RFC7950]:

- o action,
- o augment,
- o configuration data,
- o container,
- o container with presence,
- o data model,
- o data node,
- o feature,
- o leaf,
- o list,
- o mandatory node,
- o module,
- o schema tree,
- o state data,
- o RPC operation.

2.1. Glossary of New Terms

core routing data model: YANG data model comprising "ietf-routing", "ietf-ipv4-unicast-routing" and "ietf-ipv6-unicast-routing" modules.

direct route: a route to a directly connected network.

routing information base (RIB): An object containing a list of routes together with other information. See Section 5.2 for details.

system-controlled entry: An entry of a list in state data ("config false") that is created by the system independently of what has been explicitly configured. See Section 4.1 for details.

user-controlled entry: An entry of a list in state data ("config false") that is created and deleted as a direct consequence of certain configuration changes. See Section 4.1 for details.

2.2. Tree Diagrams

A simplified graphical representation of the complete data tree is presented in Appendix A, and similar diagrams of its various subtrees appear in the main text.

- o Brackets "[" and "]" enclose list keys.
- o Curly braces "{" and "}" contain names of optional features that make the corresponding node conditional.
- o Abbreviations before data node names: "rw" means configuration (read-write), "ro" state data (read-only), "-x" RPC operations or actions, and "-n" notifications.
- o Symbols after data node names: "?" means an optional node, "!" a container with presence, and "*" denotes a "list" or "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.

2.3. 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	[RFC7223]
ip	ietf-ip	[RFC7277]
rt	ietf-routing	Section 7
v4ur	ietf-ipv4-unicast-routing	Section 8
v6ur	ietf-ipv6-unicast-routing	Section 9
yang	ietf-yang-types	[RFC6991]
inet	ietf-inet-types	[RFC6991]

Table 1: Prefixes and corresponding YANG modules

3. Objectives

The initial design of the core routing data model was driven by the following objectives:

- o The data model should be suitable for the common address families, in particular IPv4 and IPv6, and for unicast and multicast routing, as well as Multiprotocol Label Switching (MPLS).
- o A simple IP routing system, such as one that uses only static routing, should be configurable in a simple way, ideally without any need to develop additional YANG modules.
- o On the other hand, the core routing framework must allow for complicated implementations involving multiple routing information bases (RIB) and multiple control plane protocols, as well as controlled redistributions of routing information.
- o Device vendors will want to map the data models built on this generic framework to their proprietary data models and configuration interfaces. Therefore, the framework should be flexible enough to facilitate such a mapping and accommodate data models with different logic.

4. The Design of the Core Routing Data Model

The core routing data model consists of three YANG modules and one submodule. The first module, "ietf-routing", defines the generic components of a routing system. The other two modules, "ietf-ipv4-unicast-routing" and "ietf-ipv6-unicast-routing", augment the "ietf-routing" module with additional data nodes that are needed for IPv4 and IPv6 unicast routing, respectively. Module "ietf-ipv6-unicast-routing" has a submodule, "ietf-ipv6-router-advertisements", that augments the "ietf-interfaces" [RFC7223] and "ietf-ip" [RFC7277] modules with configuration variables for IPv6 router advertisements as required by [RFC4861]. Figures 1 and 2 show abridged views of the configuration and state data hierarchies. See Appendix A for the complete data trees.

```

+--rw routing
  +--rw router-id?
  +--rw control-plane-protocols
  |   +--rw control-plane-protocol* [type name]
  |   |   +--rw type
  |   |   +--rw name
  |   |   +--rw description?
  |   |   +--rw static-routes
  |   |   |   +--rw v6ur:ipv6
  |   |   |   |   ...
  |   |   |   +--rw v4ur:ipv4
  |   |   |   |   ...
  |   |   |   ...
  +--rw ribs
    +--rw rib* [name]
      +--rw name
      +--rw address-family?
      +--rw description?

```

Figure 1: Configuration data hierarchy.

```

+--ro routing-state
  +--ro router-id?
  +--ro interfaces
  |   +--ro interface*
  +--ro control-plane-protocols
  |   +--ro control-plane-protocol* [type name]
  |       +--ro type
  |       +--ro name
  +--ro ribs
  |   +--ro rib* [name]
  |       +--ro name
  |       +--ro address-family
  |       +--ro default-rib?
  |       +--ro routes
  |           +--ro route*
  |           ...

```

Figure 2: State data hierarchy.

As can be seen from Figures 1 and 2, the core routing data model introduces several generic components of a routing framework: routes, RIBs containing lists of routes, and control plane protocols. Section 5 describes these components in more detail.

4.1. System-Controlled and User-Controlled List Entries

The core routing data model defines several lists in the schema tree, such as "rib", that have to be populated with at least one entry in any properly functioning device, and additional entries may be configured by a client.

In such a list, the server creates the required item as a so-called system-controlled entry in state data, i.e., inside the "routing-state" container.

An example can be seen in Appendix D: the "/routing-state/ribs/rib" list has two system-controlled entries named "ipv4-master" and "ipv6-master".

Additional entries may be created in the configuration by a client, e.g., via the NETCONF protocol. These are so-called user-controlled entries. If the server accepts a configured user-controlled entry, then this entry also appears in the state data version of the list.

Corresponding entries in both versions of the list (in state data and configuration) have the same value of the list key.

A client may also provide supplemental configuration of system-controlled entries. To do so, the client creates a new entry in the configuration with the desired contents. In order to bind this entry to the corresponding entry in the state data list, the key of the configuration entry has to be set to the same value as the key of the state entry.

Deleting a user-controlled entry from the configuration list results in the removal of the corresponding entry in the state data list. In contrast, if a system-controlled entry is deleted from the configuration list, only the extra configuration specified in that entry is removed but the corresponding state data entry remains in the list.

5. Basic Building Blocks

This section describes the essential components of the core routing data model.

5.1. Route

Routes are basic elements of information in a routing system. The core routing data model defines only the following minimal set of route attributes:

- o "destination-prefix": address prefix specifying the set of destination addresses for which the route may be used. This attribute is mandatory.
- o "route-preference": an integer value (also known as administrative distance) that is used for selecting a preferred route among routes with the same destination prefix. A lower value means a more preferred route.
- o "next-hop": determines the outgoing interface and/or next-hop address(es), other operation to be performed with a packet.

Routes are primarily state data that appear as entries of RIBs (Section 5.2) but they may also be found in configuration data, for example as manually configured static routes. In the latter case, configurable route attributes are generally a subset of attributes defined for RIB routes.

5.2. Routing Information Base (RIB)

Every implementation of the core routing data model manages one or more routing information bases (RIB). A RIB is a list of routes complemented with administrative data. Each RIB contains only routes

of one address family. An address family is represented by an identity derived from the "rt:address-family" base identity.

In the core routing data model, RIBs are state data represented as entries of the list "/routing-state/ribs/rib". The contents of RIBs are controlled and manipulated by control plane protocol operations which may result in route additions, removals and modifications. This also includes manipulations via the "static" and/or "direct" pseudo-protocols, see Section 5.3.1.

For every supported address family, exactly one RIB MUST be marked as the so-called default RIB. Its role is explained in Section 5.3.

Simple router implementations that do not advertise the feature "multiple-ribs" will typically create one system-controlled RIB per supported address family, and mark it as the default RIB.

More complex router implementations advertising the "multiple-ribs" feature support multiple RIBs per address family that can be used for policy routing and other purposes.

The following action (see Section 7.15 of [RFC7950]) is defined for the "rib" list:

- o active-route -- return the active RIB route for the destination address that is specified as the action's input parameter.

5.3. Control Plane Protocol

The core routing data model provides an open-ended framework for defining multiple control plane protocol instances, e.g., for Layer 3 routing protocols. Each control plane protocol instance MUST be assigned a type, which is an identity derived from the "rt:control-plane-protocol" base identity. The core routing data model defines two identities for the direct and static pseudo-protocols (Section 5.3.1).

Multiple control plane protocol instances of the same type MAY be configured.

5.3.1. Routing Pseudo-Protocols

The core routing data model defines two special routing protocol types -- "direct" and "static". Both are in fact pseudo-protocols, which means that they are confined to the local device and do not exchange any routing information with adjacent routers.

Every implementation of the core routing data model MUST provide exactly one instance of the "direct" pseudo-protocol type. It is the source of direct routes for all configured address families. Direct routes are normally supplied by the operating system kernel, based on the configuration of network interface addresses, see Section 6.2.

A pseudo-protocol of the type "static" allows for specifying routes manually. It MAY be configured in zero or multiple instances, although a typical configuration will have exactly one instance.

5.3.2. Defining New Control Plane Protocols

It is expected that future YANG modules will create data models for additional control plane protocol types. Such a new module has to define the protocol-specific configuration and state data, and it has to integrate it into the core routing framework in the following way:

- o A new identity MUST be defined for the control plane protocol and its base identity MUST be set to "rt:control-plane-protocol", or to an identity derived from "rt:control-plane-protocol".
- o Additional route attributes MAY be defined, preferably in one place by means of defining a YANG grouping. The new attributes have to be inserted by augmenting the definitions of the nodes

```
/rt:routing-state/rt:ribs/rt:rib/rt:routes/rt:route
```

and

```
/rt:routing-state/rt:ribs/rt:rib/rt:output/rt:route,
```

and possibly other places in the configuration, state data, notifications, and input/output parameters of actions or RPC operations.

- o Configuration parameters and/or state data for the new protocol can be defined by augmenting the "control-plane-protocol" data node under both "/routing" and "/routing-state".

By using a "when" statement, the augmented configuration parameters and state data specific to the new protocol SHOULD be made conditional and valid only if the value of "rt:type" or "rt:source-protocol" is equal to (or derived from) the new protocol's identity.

It is also RECOMMENDED that protocol-specific data nodes be encapsulated in an appropriately named container with presence. Such a container may contain mandatory data nodes that are otherwise forbidden at the top level of an augment.

The above steps are implemented by the example YANG module for the RIP routing protocol in Appendix C.

5.4. Parameters of IPv6 Router Advertisements

YANG module "ietf-ipv6-router-advertisements" (Section 9.1), which is a submodule of the "ietf-ipv6-unicast-routing" module, augments the configuration and state data of IPv6 interfaces with definitions of the following variables as required by [RFC4861], sec. 6.2.1:

- o send-advertisements,
- o max-rtr-adv-interval,
- o min-rtr-adv-interval,
- o managed-flag,
- o other-config-flag,
- o link-mtu,
- o reachable-time,
- o retrans-timer,
- o cur-hop-limit,
- o default-lifetime,
- o prefix-list: a list of prefixes to be advertised.

The following parameters are associated with each prefix in the list:

- * valid-lifetime,
- * on-link-flag,
- * preferred-lifetime,
- * autonomous-flag.

NOTES:

1. The "IsRouter" flag, which is also required by [RFC4861], is implemented in the "ietf-ip" module [RFC7277] (leaf "ip:forwarding").

2. The original specification [RFC4861] allows the implementations to decide whether the "valid-lifetime" and "preferred-lifetime" parameters remain the same in consecutive advertisements, or decrement in real time. However, the latter behavior seems problematic because the values might be reset again to the (higher) configured values after a configuration is reloaded. Moreover, no implementation is known to use the decrementing behavior. The "ietf-ipv6-router-advertisements" submodule therefore stipulates the former behavior with constant values.

6. Interactions with Other YANG Modules

The semantics of the core routing data model also depends on several configuration parameters that are defined in other YANG modules.

6.1. Module "ietf-interfaces"

The following boolean switch is defined in the "ietf-interfaces" YANG module [RFC7223]:

```
/if:interfaces/if:interface/if:enabled
```

If this switch is set to "false" for a network layer interface, then all routing and forwarding functions MUST be disabled on that interface.

6.2. Module "ietf-ip"

The following boolean switches are defined in the "ietf-ip" YANG module [RFC7277]:

```
/if:interfaces/if:interface/ip:ipv4/ip:enabled
```

If this switch is set to "false" for a network layer interface, then all IPv4 routing and forwarding functions MUST be disabled on that interface.

```
/if:interfaces/if:interface/ip:ipv4/ip:forwarding
```

If this switch is set to "false" for a network layer interface, then the forwarding of IPv4 datagrams through this interface MUST be disabled. However, the interface MAY participate in other IPv4 routing functions, such as routing protocols.

```
/if:interfaces/if:interface/ip:ipv6/ip:enabled
```

If this switch is set to "false" for a network layer interface, then all IPv6 routing and forwarding functions MUST be disabled on that interface.

```
/if:interfaces/if:interface/ip:ipv6/ip:forwarding
```

If this switch is set to "false" for a network layer interface, then the forwarding of IPv6 datagrams through this interface MUST be disabled. However, the interface MAY participate in other IPv6 routing functions, such as routing protocols.

In addition, the "ietf-ip" module allows for configuring IPv4 and IPv6 addresses and network prefixes or masks on network layer interfaces. Configuration of these parameters on an enabled interface MUST result in an immediate creation of the corresponding direct route. The destination prefix of this route is set according to the configured IP address and network prefix/mask, and the interface is set as the outgoing interface for that route.

7. Routing Management YANG Module

RFC Editor: In this section, replace all occurrences of 'XXXX' with the actual RFC number and all occurrences of the revision date below with the date of RFC publication (and remove this note).

```
<CODE BEGINS> file "ietf-routing@2016-11-03.yang"

module ietf-routing {

  yang-version "1.1";

  namespace "urn:ietf:params:xml:ns:yang:ietf-routing";

  prefix "rt";

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

  import ietf-interfaces {
    prefix "if";
  }

  organization
    "IETF NETMOD (NETCONF Data Modeling Language) Working Group";

  contact
    "WG Web: <https://tools.ietf.org/wg/netmod/>
```

WG List: <mailto:netmod@ietf.org>
WG Chair: Lou Berger
<mailto:lberger@labn.net>
WG Chair: Kent Watsen
<mailto:kwatsen@juniper.net>
Editor: Ladislav Lhotka
<mailto:lhotka@nic.cz>
Editor: Acee Lindem
<mailto:acee@cisco.com>;

description

"This YANG module defines essential components for the management of a routing subsystem.

Copyright (c) 2016 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 (<https://trustee.ietf.org/license-info>).

The key words 'MUST', 'MUST NOT', 'REQUIRED', 'SHALL', 'SHALL NOT', 'SHOULD', 'SHOULD NOT', 'RECOMMENDED', 'MAY', and 'OPTIONAL' in the module text are to be interpreted as described in RFC 2119 (<https://tools.ietf.org/html/rfc2119>).

This version of this YANG module is part of RFC XXXX (<https://tools.ietf.org/html/rfcXXXX>); see the RFC itself for full legal notices.";

```
revision 2016-11-03 {  
  description  
    "Initial revision."  
  reference  
    "RFC XXXX: A YANG Data Model for Routing Management";  
}
```

```
/* Features */
```

```
feature multiple-ribs {  
  description
```

```
"This feature indicates that the server supports user-defined
RIBs.

Servers that do not advertise this feature SHOULD provide
exactly one system-controlled RIB per supported address family
and make them also the default RIBs. These RIBs then appear as
entries of the list /routing-state/ribs/rib.";
}

feature router-id {
  description
    "This feature indicates that the server supports configuration
    of an explicit 32-bit router ID that is used by some routing
    protocols.

    Servers that do not advertise this feature set a router ID
    algorithmically, usually to one of configured IPv4 addresses.
    However, this algorithm is implementation-specific.";
}

/* Identities */

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.";
}

identity control-plane-protocol {
  description
    "Base identity from which control plane protocol identities are
    derived.";
}

identity routing-protocol {
  base control-plane-protocol;
}
```

```
    description
      "Identity from which Layer 3 routing protocol identities are
      derived.";
  }

  identity direct {
    base routing-protocol;
    description
      "Routing pseudo-protocol that provides routes to directly
      connected networks.";
  }

  identity static {
    base routing-protocol;
    description
      "Static routing pseudo-protocol.";
  }

  /* Type Definitions */

  typedef route-preference {
    type uint32;
    description
      "This type is used for route preferences.";
  }

  /* Groupings */

  grouping address-family {
    description
      "This grouping provides a leaf identifying an address
      family.";
    leaf address-family {
      type identityref {
        base address-family;
      }
      mandatory "true";
      description
        "Address family.";
    }
  }

  grouping router-id {
    description
      "This grouping provides router ID.";
    leaf router-id {
      type yang:dotted-quad;
      description

```

```
        "A 32-bit number in the form of a dotted quad that is used by
        some routing protocols identifying a router.";
    reference
        "RFC 2328: OSPF Version 2.";
    }
}

grouping special-next-hop {
    description
        "This grouping provides a leaf with an enumeration of special
        next-hops.";
    leaf special-next-hop {
        type enumeration {
            enum blackhole {
                description
                    "Silently discard the packet.";
            }
            enum unreachable {
                description
                    "Discard the packet and notify the sender with an error
                    message indicating that the destination host is
                    unreachable.";
            }
            enum prohibit {
                description
                    "Discard the packet and notify the sender with an error
                    message indicating that the communication is
                    administratively prohibited.";
            }
            enum receive {
                description
                    "The packet will be received by the local system.";
            }
        }
    }
    description
        "Special next-hop options.";
}

grouping next-hop-content {
    description
        "Generic parameters of next-hops in static routes.";
    choice next-hop-options {
        mandatory "true";
        description
            "Options for next-hops in static routes.
```

It is expected that further cases will be added through

```
    augments from other modules.";
case simple-next-hop {
  description
    "This case represents a simple next hop consisting of the
    next-hop address and/or outgoing interface.

    Modules for address families MUST augment this case with a
    leaf containing a next-hop address of that address
    family.";
  leaf outgoing-interface {
    type if:interface-ref;
    description
      "Name of the outgoing interface.";
  }
}
case special-next-hop {
  uses special-next-hop;
}
case next-hop-list {
  container next-hop-list {
    description
      "Container for multiple next-hops.";
    list next-hop {
      key "index";
      description
        "An entry of a next-hop list.

        Modules for address families MUST augment this list
        with a leaf containing a next-hop address of that
        address family.";
      leaf index {
        type string;
        description
          "An user-specified identifier utilised to uniquely
          reference the next-hop entry in the next-hop list.
          The value of this index has no semantic meaning
          other than for referencing the entry.";
      }
      leaf outgoing-interface {
        type if:interface-ref;
        description
          "Name of the outgoing interface.";
      }
    }
  }
}
}
```

```
grouping next-hop-state-content {
  description
    "Generic parameters of next-hops in state data.";
  choice next-hop-options {
    mandatory "true";
    description
      "Options for next-hops in state data.

      It is expected that further cases will be added through
      augments from other modules, e.g., for recursive
      next-hops.";
    case simple-next-hop {
      description
        "This case represents a simple next hop consisting of the
        next-hop address and/or outgoing interface.

        Modules for address families MUST augment this case with a
        leaf containing a next-hop address of that address
        family.";
      leaf outgoing-interface {
        type if:interface-state-ref;
        description
          "Name of the outgoing interface.";
      }
    }
    case special-next-hop {
      uses special-next-hop;
    }
    case next-hop-list {
      container next-hop-list {
        description
          "Container for multiple next-hops.";
        list next-hop {
          description
            "An entry of a next-hop list.

            Modules for address families MUST augment this list
            with a leaf containing a next-hop address of that
            address family.";
          leaf outgoing-interface {
            type if:interface-state-ref;
            description
              "Name of the outgoing interface.";
          }
        }
      }
    }
  }
}
```

```
    }

    grouping route-metadata {
      description
        "Common route metadata.";
      leaf source-protocol {
        type identityref {
          base routing-protocol;
        }
        mandatory "true";
        description
          "Type of the routing protocol from which the route
           originated.";
      }
      leaf active {
        type empty;
        description
          "Presence of this leaf indicates that the route is preferred
           among all routes in the same RIB that have the same
           destination prefix.";
      }
      leaf last-updated {
        type yang:date-and-time;
        description
          "Time stamp of the last modification of the route. If the
           route was never modified, it is the time when the route was
           inserted into the RIB.";
      }
    }
  }
}

/* State data */

container routing-state {
  config "false";
  description
    "State data of the routing subsystem.";
  uses router-id {
    description
      "Global router ID.

       It may be either configured or assigned algorithmically by
       the implementation.";
  }
  container interfaces {
    description
      "Network layer interfaces used for routing.";
    leaf-list interface {
      type if:interface-state-ref;
    }
  }
}
```

```
        description
            "Each entry is a reference to the name of a configured
            network layer interface.";
    }
}
container control-plane-protocols {
    description
        "Container for the list of routing protocol instances.";
    list control-plane-protocol {
        key "type name";
        description
            "State data of a control plane protocol instance.

            An implementation MUST provide exactly one
            system-controlled instance of the 'direct'
            pseudo-protocol. Instances of other control plane
            protocols MAY be created by configuration.";
        leaf type {
            type identityref {
                base control-plane-protocol;
            }
            description
                "Type of the control plane protocol.";
        }
        leaf name {
            type string;
            description
                "The name of the control plane protocol instance.

                For system-controlled instances this name is persistent,
                i.e., it SHOULD NOT change across reboots.";
        }
    }
}
container ribs {
    description
        "Container for RIBs.";
    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.

            An implementation SHOULD provide one system-controlled
            default RIB for each supported address family.";
        leaf name {
```

```
    type string;
    description
      "The name of the RIB.";
  }
  uses address-family;
  leaf default-rib {
    if-feature "multiple-ribs";
    type boolean;
    default "true";
    description
      "This flag has the value of 'true' if and only if the RIB
      is the default RIB for the given address family.

      By default, control plane protocols place their routes
      in the default RIBs.";
  }
  container routes {
    description
      "Current content of the RIB.";
    list route {
      description
        "A RIB route entry. This data node MUST be augmented
        with information specific for routes of each address
        family.";
      leaf route-preference {
        type route-preference;
        description
          "This route attribute, also known as administrative
          distance, allows for selecting the preferred route
          among routes with the same destination prefix. A
          smaller value means a more preferred route.";
      }
      container next-hop {
        description
          "Route's next-hop attribute.";
        uses next-hop-state-content;
      }
      uses route-metadata;
    }
  }
  action active-route {
    description
      "Return the active RIB route that is used for the
      destination address.

      Address family specific modules MUST augment input
      parameters with a leaf named 'destination-address'.";
    output {
```

```

    container route {
      description
        "The active RIB route for the specified destination.

        If no route exists in the RIB for the destination
        address, no output is returned.

        Address family specific modules MUST augment this
        container with appropriate route contents.";
      container next-hop {
        description
          "Route's next-hop attribute.";
        uses next-hop-state-content;
      }
      uses route-metadata;
    }
  }
}

/* Configuration Data */

container routing {
  description
    "Configuration parameters for the routing subsystem.";
  uses router-id {
    if-feature "router-id";
    description
      "Configuration of the global router ID. Routing protocols
      that use router ID can use this parameter or override it
      with another value.";
  }
  container control-plane-protocols {
    description
      "Configuration of control plane protocol instances.";
    list control-plane-protocol {
      key "type name";
      description
        "Each entry contains configuration of a control plane
        protocol instance.";
      leaf type {
        type identityref {
          base control-plane-protocol;
        }
        description
          "Type of the control plane protocol - an identity derived

```



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

organization
  "IETF NETMOD (NETCONF Data Modeling Language) Working Group";

contact
  "WG Web: <https://tools.ietf.org/wg/netmod/>
  WG List: <mailto:netmod@ietf.org>

  WG Chair: Lou Berger
            <mailto:lberger@labn.net>

  WG Chair: Kent Watsen
            <mailto:kwatsen@juniper.net>

  Editor:   Ladislav Lhotka
            <mailto:lhotka@nic.cz>

  Editor:   Acee Lindem
            <mailto:acee@cisco.com>";

description
  "This YANG module augments the 'ietf-routing' module with basic
  configuration and state data for IPv4 unicast routing.

  Copyright (c) 2016 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
  (https://trustee.ietf.org/license-info).

  The key words 'MUST', 'MUST NOT', 'REQUIRED', 'SHALL', 'SHALL
  NOT', 'SHOULD', 'SHOULD NOT', 'RECOMMENDED', 'MAY', and
  'OPTIONAL' in the module text are to be interpreted as described
  in RFC 2119 (https://tools.ietf.org/html/rfc2119).

  This version of this YANG module is part of RFC XXXX
  (https://tools.ietf.org/html/rfcXXXX); see the RFC itself for
  full legal notices.";

revision 2016-11-03 {
  description
```

```
    "Initial revision.";
  reference
    "RFC XXXX: A YANG Data Model for Routing Management";
}

/* Identities */

identity ipv4-unicast {
  base rt:ipv4;
  description
    "This identity represents the IPv4 unicast address family.";
}

/* State data */

augment "/rt:routing-state/rt:ribs/rt:rib/rt:routes/rt:route" {
  when "derived-from-or-self(..../rt:address-family, "
    + "'v4ur:ipv4-unicast')" {
    description
      "This augment is valid only for IPv4 unicast.";
  }
  description
    "This leaf augments an IPv4 unicast route.";
  leaf destination-prefix {
    type inet:ipv4-prefix;
    description
      "IPv4 destination prefix.";
  }
}

augment "/rt:routing-state/rt:ribs/rt:rib/rt:routes/rt:route/"
  + "rt:next-hop/rt:next-hop-options/rt:simple-next-hop" {
  when "derived-from-or-self(..../..../rt:address-family, "
    + "'v4ur:ipv4-unicast')" {
    description
      "This augment is valid only for IPv4 unicast.";
  }
  description
    "Augment 'simple-next-hop' case in IPv4 unicast routes.";
  leaf next-hop-address {
    type inet:ipv4-address;
    description
      "IPv4 address of the next-hop.";
  }
}

augment "/rt:routing-state/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" {
when "derived-from-or-self(..../rt:address-family, "
+ "'v4ur:ipv4-unicast')" {
  description
    "This augment is valid only for IPv4 unicast.";
}
description
  "This leaf augments the 'next-hop-list' case of IPv4 unicast
  routes.";
leaf address {
  type inet:ipv4-address;
  description
    "IPv4 address of the next-hop.";
}
}

augment
  "/rt:routing-state/rt:ribs/rt:rib/rt:active-route/rt:input" {
when "derived-from-or-self(..../rt:address-family, "
+ "'v4ur:ipv4-unicast')" {
  description
    "This augment is valid only for IPv4 unicast RIBs.";
}
description
  "This augment adds the input parameter of the 'active-route'
  action.";
leaf destination-address {
  type inet:ipv4-address;
  description
    "IPv4 destination address.";
}
}

augment "/rt:routing-state/rt:ribs/rt:rib/rt:active-route/"
  + "rt:output/rt:route" {
when "derived-from-or-self(..../rt:address-family, "
+ "'v4ur:ipv4-unicast')" {
  description
    "This augment is valid only for IPv4 unicast.";
}
description
  "This augment adds the destination prefix to the reply of the
  'active-route' action.";
leaf destination-prefix {
  type inet:ipv4-prefix;
  description
    "IPv4 destination prefix.";
}
}
```

```
    }

    augment "/rt:routing-state/rt:ribs/rt:rib/rt:active-route/"
      + "rt:output/rt:route/rt:next-hop/rt:next-hop-options/"
      + "rt:simple-next-hop" {
    when "derived-from-or-self(..../..../rt:address-family, "
      + "'v4ur:ipv4-unicast')" {
      description
        "This augment is valid only for IPv4 unicast.";
    }
    description
      "Augment 'simple-next-hop' case in the reply to the
      'active-route' action.";
    leaf next-hop-address {
      type inet:ipv4-address;
      description
        "IPv4 address of the next-hop.";
    }
  }
}

augment "/rt:routing-state/rt:ribs/rt:rib/rt:active-route/"
  + "rt:output/rt:route/rt:next-hop/rt:next-hop-options/"
  + "rt:next-hop-list/rt:next-hop-list/rt:next-hop" {
when "derived-from-or-self(..../..../..../rt:address-family, "
  + "'v4ur:ipv4-unicast')" {
  description
    "This augment is valid only for IPv4 unicast.";
}
description
  "Augment 'next-hop-list' case in the reply to the
  'active-route' action.";
leaf next-hop-address {
  type inet:ipv4-address;
  description
    "IPv4 address of the next-hop.";
}
}

/* Configuration data */

augment "/rt:routing/rt:control-plane-protocols/"
  + "rt:control-plane-protocol/rt:static-routes" {
description
  "This augment defines the configuration of the 'static'
  pseudo-protocol with data specific to IPv4 unicast.";
container ipv4 {
description
  "Configuration of a 'static' pseudo-protocol instance
```


<CODE ENDS>

9. IPv6 Unicast Routing Management YANG Module

RFC Editor: In this section, replace all occurrences of 'XXXX' with the actual RFC number and all occurrences of the revision date below with the date of RFC publication (and remove this note).

```
<CODE BEGINS> file "ietf-ipv6-unicast-routing@2016-11-03.yang"

module ietf-ipv6-unicast-routing {

  yang-version "1.1";

  namespace "urn:ietf:params:xml:ns:yang:ietf-ipv6-unicast-routing";

  prefix "v6ur";

  import ietf-routing {
    prefix "rt";
  }

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

  include ietf-ipv6-router-advertisements {
    revision-date 2016-11-03;
  }

  organization
    "IETF NETMOD (NETCONF Data Modeling Language) Working Group";

  contact
    "WG Web: <https://tools.ietf.org/wg/netmod/>
    WG List: <mailto:netmod@ietf.org>

    WG Chair: Lou Berger
              <mailto:lberger@labn.net>

    WG Chair: Kent Watsen
              <mailto:kwatsen@juniper.net>

    Editor: Ladislav Lhotka
            <mailto:lhotka@nic.cz>

    Editor: Acee Lindem
            <mailto:acee@cisco.com>";
```

description

"This YANG module augments the 'ietf-routing' module with basic configuration and state data for IPv6 unicast routing.

Copyright (c) 2016 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 (<https://trustee.ietf.org/license-info>).

The key words 'MUST', 'MUST NOT', 'REQUIRED', 'SHALL', 'SHALL NOT', 'SHOULD', 'SHOULD NOT', 'RECOMMENDED', 'MAY', and 'OPTIONAL' in the module text are to be interpreted as described in RFC 2119 (<https://tools.ietf.org/html/rfc2119>).

This version of this YANG module is part of RFC XXXX (<https://tools.ietf.org/html/rfcXXXX>); see the RFC itself for full legal notices."

```
revision 2016-11-03 {
  description
    "Initial revision.";
  reference
    "RFC XXXX: A YANG Data Model for Routing Management";
}

/* Identities */

identity ipv6-unicast {
  base rt:ipv6;
  description
    "This identity represents the IPv6 unicast address family.";
}

/* State data */

augment "/rt:routing-state/rt:ribs/rt:rib/rt:routes/rt:route" {
  when "derived-from-or-self(..../rt:address-family, "
    + "'v6ur:ipv6-unicast')" {
    description
      "This augment is valid only for IPv6 unicast.";
  }
  description
    "This leaf augments an IPv6 unicast route.";
```

```

    leaf destination-prefix {
      type inet:ipv6-prefix;
      description
        "IPv6 destination prefix.";
    }
  }

  augment "/rt:routing-state/rt:ribs/rt:rib/rt:routes/rt:route/"
    + "rt:next-hop/rt:next-hop-options/rt:simple-next-hop" {
    when "derived-from-or-self(..../rt:address-family, "
      + "'v6ur:ipv6-unicast')" {
      description
        "This augment is valid only for IPv6 unicast.";
    }
    description
      "Augment 'simple-next-hop' case in IPv6 unicast routes.";
    leaf next-hop-address {
      type inet:ipv6-address;
      description
        "IPv6 address of the next-hop.";
    }
  }

  augment "/rt:routing-state/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" {
    when "derived-from-or-self(..../rt:address-family, "
      + "'v6ur:ipv6-unicast')" {
      description
        "This augment is valid only for IPv6 unicast.";
    }
    description
      "This leaf augments the 'next-hop-list' case of IPv6 unicast
      routes.";
    leaf address {
      type inet:ipv6-address;
      description
        "IPv6 address of the next-hop.";
    }
  }

  augment
    "/rt:routing-state/rt:ribs/rt:rib/rt:active-route/rt:input" {
    when "derived-from-or-self(../rt:address-family, "
      + "'v6ur:ipv6-unicast')" {
      description
        "This augment is valid only for IPv6 unicast RIBs.";
    }
  }

```

```
description
  "This augment adds the input parameter of the 'active-route'
  action.";
leaf destination-address {
  type inet:ipv6-address;
  description
    "IPv6 destination address.";
}
}

augment "/rt:routing-state/rt:ribs/rt:rib/rt:active-route/"
  + "rt:output/rt:route" {
  when "derived-from-or-self(..../rt:address-family, "
    + "'v6ur:ipv6-unicast')" {
    description
      "This augment is valid only for IPv6 unicast.";
  }
  description
    "This augment adds the destination prefix to the reply of the
    'active-route' action.";
  leaf destination-prefix {
    type inet:ipv6-prefix;
    description
      "IPv6 destination prefix.";
  }
}

augment "/rt:routing-state/rt:ribs/rt:rib/rt:active-route/"
  + "rt:output/rt:route/rt:next-hop/rt:next-hop-options/"
  + "rt:simple-next-hop" {
  when "derived-from-or-self(..../rt:address-family, "
    + "'v6ur:ipv6-unicast')" {
    description
      "This augment is valid only for IPv6 unicast.";
  }
  description
    "Augment 'simple-next-hop' case in the reply to the
    'active-route' action.";
  leaf next-hop-address {
    type inet:ipv6-address;
    description
      "IPv6 address of the next-hop.";
  }
}

augment "/rt:routing-state/rt:ribs/rt:rib/rt:active-route/"
  + "rt:output/rt:route/rt:next-hop/rt:next-hop-options/"
  + "rt:next-hop-list/rt:next-hop-list/rt:next-hop" {
```

```
when "derived-from-or-self(..../..../rt:address-family, "
  + "'v6ur:ipv6-unicast')" {
  description
    "This augment is valid only for IPv6 unicast.";
}
description
  "Augment 'next-hop-list' case in the reply to the
  'active-route' action.";
leaf next-hop-address {
  type inet:ipv6-address;
  description
    "IPv6 address of the next-hop.";
}
}
}

/* Configuration data */

augment "/rt:routing/rt:control-plane-protocols/"
  + "rt:control-plane-protocol/rt:static-routes" {
  description
    "This augment defines the configuration of the 'static'
    pseudo-protocol with data specific to IPv6 unicast.";
  container ipv6 {
    description
      "Configuration of a 'static' pseudo-protocol instance
      consists of a list of routes.";
    list route {
      key "destination-prefix";
      description
        "A list of static routes.";
      leaf destination-prefix {
        type inet:ipv6-prefix;
        mandatory "true";
        description
          "IPv6 destination prefix.";
      }
      leaf description {
        type string;
        description
          "Textual description of the route.";
      }
    }
    container next-hop {
      description
        "Configuration of next-hop.";
      uses rt:next-hop-content {
        augment "next-hop-options/simple-next-hop" {
          description
            "Augment 'simple-next-hop' case in IPv6 static
```



```
    prefix "if";
  }

  import ietf-ip {
    prefix "ip";
  }

  organization
    "IETF NETMOD (NETCONF Data Modeling Language) Working Group";

  contact
    "WG Web:    <https://tools.ietf.org/wg/netmod/>
     WG List:   <mailto:netmod@ietf.org>

     WG Chair:  Lou Berger
                <mailto:lberger@labn.net>

     WG Chair:  Kent Watsen
                <mailto:kwatsen@juniper.net>

     Editor:    Ladislav Lhotka
                <mailto:lhotka@nic.cz>

     Editor:    Acee Lindem
                <mailto:acee@cisco.com>";

  description
    "This YANG module augments the 'ietf-ip' module with
     configuration and state data of IPv6 router advertisements.

     Copyright (c) 2016 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
     (https://trustee.ietf.org/license-info).

     The key words 'MUST', 'MUST NOT', 'REQUIRED', 'SHALL', 'SHALL
     NOT', 'SHOULD', 'SHOULD NOT', 'RECOMMENDED', 'MAY', and
     'OPTIONAL' in the module text are to be interpreted as described
     in RFC 2119 (https://tools.ietf.org/html/rfc2119).

     This version of this YANG module is part of RFC XXXX
     (https://tools.ietf.org/html/rfcXXXX); see the RFC itself for
     full legal notices.";
```

```
reference
  "RFC 4861: Neighbor Discovery for IP version 6 (IPv6).";

revision 2016-11-03 {
  description
    "Initial revision.";
  reference
    "RFC XXXX: A YANG Data Model for Routing Management";
}

/* State data */

augment "/if:interfaces-state/if:interface/ip:ipv6" {
  description
    "Augment interface state data with parameters of IPv6 router
    advertisements.";
  container ipv6-router-advertisements {
    description
      "Parameters of IPv6 Router Advertisements.";
    leaf send-advertisements {
      type boolean;
      description
        "A flag indicating whether or not the router sends periodic
        Router Advertisements and responds to Router
        Solicitations.";
    }
    leaf max-rtr-adv-interval {
      type uint16 {
        range "4..1800";
      }
      units "seconds";
      description
        "The maximum time allowed between sending unsolicited
        multicast Router Advertisements from the interface.";
    }
    leaf min-rtr-adv-interval {
      type uint16 {
        range "3..1350";
      }
      units "seconds";
      description
        "The minimum time allowed between sending unsolicited
        multicast Router Advertisements from the interface.";
    }
    leaf managed-flag {
      type boolean;
      description
        "The value that is placed in the 'Managed address
```

```
        configuration' flag field in the Router Advertisement.";
    }
    leaf other-config-flag {
        type boolean;
        description
            "The value that is placed in the 'Other configuration' flag
            field in the Router Advertisement.";
    }
    leaf link-mtu {
        type uint32;
        description
            "The value that is placed in MTU options sent by the
            router. A value of zero indicates that no MTU options are
            sent.";
    }
    leaf reachable-time {
        type uint32 {
            range "0..3600000";
        }
        units "milliseconds";
        description
            "The value that is placed in the Reachable Time field in
            the Router Advertisement messages sent by the router. A
            value of zero means unspecified (by this router).";
    }
    leaf retrans-timer {
        type uint32;
        units "milliseconds";
        description
            "The value that is placed in the Retrans Timer field in the
            Router Advertisement messages sent by the router. A value
            of zero means unspecified (by this router).";
    }
    leaf cur-hop-limit {
        type uint8;
        description
            "The value that is placed in the Cur Hop Limit field in the
            Router Advertisement messages sent by the router. A value
            of zero means unspecified (by this router).";
    }
    leaf default-lifetime {
        type uint16 {
            range "0..9000";
        }
        units "seconds";
        description
            "The value that is placed in the Router Lifetime field of
            Router Advertisements sent from the interface, in seconds.
```

```
        A value of zero indicates that the router is not to be
        used as a default router.";
    }
    container prefix-list {
        description
            "A list of prefixes that are placed in Prefix Information
            options in Router Advertisement messages sent from the
            interface.

            By default, these are all prefixes that the router
            advertises via routing protocols as being on-link for the
            interface from which the advertisement is sent.";
        list prefix {
            key "prefix-spec";
            description
                "Advertised prefix entry and its parameters.";
            leaf prefix-spec {
                type inet:ipv6-prefix;
                description
                    "IPv6 address prefix.";
            }
            leaf valid-lifetime {
                type uint32;
                units "seconds";
                description
                    "The value that is placed in the Valid Lifetime in the
                    Prefix Information option. The designated value of all
                    1's (0xffffffff) represents infinity.

                    An implementation SHOULD keep this value constant in
                    consecutive advertisements except when it is
                    explicitly changed in configuration.";
            }
            leaf on-link-flag {
                type boolean;
                description
                    "The value that is placed in the on-link flag ('L-bit')
                    field in the Prefix Information option.";
            }
            leaf preferred-lifetime {
                type uint32;
                units "seconds";
                description
                    "The value that is placed in the Preferred Lifetime in
                    the Prefix Information option, in seconds. The
                    designated value of all 1's (0xffffffff) represents
                    infinity.
```

```
        An implementation SHOULD keep this value constant in
        consecutive advertisements except when it is
        explicitly changed in configuration.";
    }
    leaf autonomous-flag {
        type boolean;
        description
            "The value that is placed in the Autonomous Flag field
            in the Prefix Information option.";
    }
}
}
}
}
}

/* Configuration data */

augment "/if:interfaces/if:interface/ip:ipv6" {
    description
        "Augment interface configuration with parameters of IPv6 router
        advertisements.";
    container ipv6-router-advertisements {
        description
            "Configuration of IPv6 Router Advertisements.";
        leaf send-advertisements {
            type boolean;
            default "false";
            description
                "A flag indicating whether or not the router sends periodic
                Router Advertisements and responds to Router
                Solicitations.";
            reference
                "RFC 4861: Neighbor Discovery for IP version 6 (IPv6) -
                AdvSendAdvertisements.";
        }
        leaf max-rtr-adv-interval {
            type uint16 {
                range "4..1800";
            }
            units "seconds";
            default "600";
            description
                "The maximum time allowed between sending unsolicited
                multicast Router Advertisements from the interface.";
            reference
                "RFC 4861: Neighbor Discovery for IP version 6 (IPv6) -
                MaxRtrAdvInterval.";
        }
    }
}
```

```
leaf min-rtr-adv-interval {
  type uint16 {
    range "3..1350";
  }
  units "seconds";
  must ". <= 0.75 * ../max-rtr-adv-interval" {
    description
      "The value MUST NOT be greater than 75 % of
       'max-rtr-adv-interval'.";
  }
  description
    "The minimum time allowed between sending unsolicited
     multicast Router Advertisements from the interface.

     The default value to be used operationally if this leaf is
     not configured is determined as follows:

     - if max-rtr-adv-interval >= 9 seconds, the default value
       is 0.33 * max-rtr-adv-interval;

     - otherwise it is 0.75 * max-rtr-adv-interval.";
  reference
    "RFC 4861: Neighbor Discovery for IP version 6 (IPv6) -
     MinRtrAdvInterval.";
}
leaf managed-flag {
  type boolean;
  default "false";
  description
    "The value to be placed in the 'Managed address
     configuration' flag field in the Router Advertisement.";
  reference
    "RFC 4861: Neighbor Discovery for IP version 6 (IPv6) -
     AdvManagedFlag.";
}
leaf other-config-flag {
  type boolean;
  default "false";
  description
    "The value to be placed in the 'Other configuration' flag
     field in the Router Advertisement.";
  reference
    "RFC 4861: Neighbor Discovery for IP version 6 (IPv6) -
     AdvOtherConfigFlag.";
}
leaf link-mtu {
  type uint32;
  default "0";
}
```

```
description
  "The value to be placed in MTU options sent by the router.
  A value of zero indicates that no MTU options are sent.";
reference
  "RFC 4861: Neighbor Discovery for IP version 6 (IPv6) -
  AdvLinkMTU.";
}
leaf reachable-time {
  type uint32 {
    range "0..3600000";
  }
  units "milliseconds";
  default "0";
  description
    "The value to be placed in the Reachable Time field in the
    Router Advertisement messages sent by the router. A value
    of zero means unspecified (by this router).";
  reference
    "RFC 4861: Neighbor Discovery for IP version 6 (IPv6) -
    AdvReachableTime.";
}
leaf retrans-timer {
  type uint32;
  units "milliseconds";
  default "0";
  description
    "The value to be placed in the Retrans Timer field in the
    Router Advertisement messages sent by the router. A value
    of zero means unspecified (by this router).";
  reference
    "RFC 4861: Neighbor Discovery for IP version 6 (IPv6) -
    AdvRetransTimer.";
}
leaf cur-hop-limit {
  type uint8;
  description
    "The value to be placed in the Cur Hop Limit field in the
    Router Advertisement messages sent by the router. A value
    of zero means unspecified (by this router).

    If this parameter is not configured, the device SHOULD use
    the value specified in IANA Assigned Numbers that was in
    effect at the time of implementation.";
  reference
    "RFC 4861: Neighbor Discovery for IP version 6 (IPv6) -
    AdvCurHopLimit.

    IANA: IP Parameters,
```

```
        http://www.iana.org/assignments/ip-parameters";
    }
leaf default-lifetime {
    type uint16 {
        range "0..9000";
    }
    units "seconds";
    description
        "The value to be placed in the Router Lifetime field of
        Router Advertisements sent from the interface, in seconds.
        It MUST be either zero or between max-rtr-adv-interval and
        9000 seconds. A value of zero indicates that the router is
        not to be used as a default router. These limits may be
        overridden by specific documents that describe how IPv6
        operates over different link layers.

        If this parameter is not configured, the device SHOULD use
        a value of 3 * max-rtr-adv-interval.";
    reference
        "RFC 4861: Neighbor Discovery for IP version 6 (IPv6) -
        AdvDefaultLifeTime.";
}
container prefix-list {
    description
        "Configuration of prefixes to be placed in Prefix
        Information options in Router Advertisement messages sent
        from the interface.

        Prefixes that are advertised by default but do not have
        their entries in the child 'prefix' list are advertised
        with the default values of all parameters.

        The link-local prefix SHOULD NOT be included in the list
        of advertised prefixes.";
    reference
        "RFC 4861: Neighbor Discovery for IP version 6 (IPv6) -
        AdvPrefixList.";
    list prefix {
        key "prefix-spec";
        description
            "Configuration of an advertised prefix entry.";
        leaf prefix-spec {
            type inet:ipv6-prefix;
            description
                "IPv6 address prefix.";
        }
        choice control-adv-prefixes {
            default "advertise";
        }
    }
}
```

```
description
  "The prefix either may be explicitly removed from the
   set of advertised prefixes, or parameters with which
   it is advertised may be specified (default case).";
leaf no-advertise {
  type empty;
  description
    "The prefix will not be advertised.

    This can be used for removing the prefix from the
    default set of advertised prefixes.";
}
case advertise {
  leaf valid-lifetime {
    type uint32;
    units "seconds";
    default "2592000";
    description
      "The value to be placed in the Valid Lifetime in
       the Prefix Information option. The designated
       value of all 1's (0xffffffff) represents
       infinity.";
    reference
      "RFC 4861: Neighbor Discovery for IP version 6
       (IPv6) - AdvValidLifetime.";
  }
  leaf on-link-flag {
    type boolean;
    default "true";
    description
      "The value to be placed in the on-link flag
       ('L-bit') field in the Prefix Information
       option.";
    reference
      "RFC 4861: Neighbor Discovery for IP version 6
       (IPv6) - AdvOnLinkFlag.";
  }
  leaf preferred-lifetime {
    type uint32;
    units "seconds";
    must ". <= ../valid-lifetime" {
      description
        "This value MUST NOT be greater than
         valid-lifetime.";
    }
    default "604800";
    description
      "The value to be placed in the Preferred Lifetime
```

URI: urn:ietf:params:xml:ns:yang:ietf-ipv4-unicast-routing

Registrant Contact: The IESG.

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

URI: urn:ietf:params:xml:ns:yang:ietf-ipv6-unicast-routing

Registrant Contact: The IESG.

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

This document registers the following YANG modules in the YANG Module Names registry [RFC6020]:

name: ietf-routing
namespace: urn:ietf:params:xml:ns:yang:ietf-routing
prefix: rt
reference: RFC XXXX

name: ietf-ipv4-unicast-routing
namespace: urn:ietf:params:xml:ns:yang:ietf-ipv4-unicast-routing
prefix: v4ur
reference: RFC XXXX

name: ietf-ipv6-unicast-routing
namespace: urn:ietf:params:xml:ns:yang:ietf-ipv6-unicast-routing
prefix: v6ur
reference: RFC XXXX

This document registers the following YANG submodule in the YANG Module Names registry [RFC6020]:

name: ietf-ipv6-router-advertisements
parent: ietf-ipv6-unicast-routing
reference: RFC XXXX

11. Security Considerations

Configuration and state data conforming to the core routing data model (defined in this document) are designed to be accessed via a management protocol with secure transport layer, such as NETCONF [RFC6241]. 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.

A number of configuration data nodes defined in the YANG modules belonging to the core routing data model are writable/creatable/deletable (i.e., "config true" in YANG terms, which is the default). These data nodes may be considered sensitive or vulnerable in some network environments. Write operations to these data nodes, such as "edit-config" in NETCONF, can have negative effects on the network if the protocol operations are not properly protected.

The vulnerable "config true" parameters and subtrees are the following:

`/routing/control-plane-protocols/control-plane-protocol:` This list specifies the control plane protocols configured on a device.

`/routing/ribs/rib:` This list specifies the RIBs configured for the device.

Unauthorised access to any of these lists can adversely affect the routing subsystem of both the local device and the network. This may lead to network malfunctions, delivery of packets to inappropriate destinations and other problems.

12. Acknowledgments

The authors wish to thank Nitin Bahadur, Martin Bjorklund, Dean Bogdanovic, Jeff Haas, Joel Halpern, Wes Hardaker, Sriganesh Kini, David Lamparter, Andrew McGregor, Jan Medved, Xiang Li, Stephane Litkowski, Thomas Morin, Tom Petch, Yingzhen Qu, Bruno Rijsman, Juergen Schoenwaelder, Phil Shafer, Dave Thaler, Yi Yang, Derek Man-Kit Yeung and Jeffrey Zhang for their helpful comments and suggestions.

13. References

13.1. 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>>.
- [RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <<http://www.rfc-editor.org/info/rfc3688>>.
- [RFC4861] Narten, T., Nordmark, E., Simpson, W., and H. Soliman, "Neighbor Discovery for IP version 6 (IPv6)", RFC 4861, DOI 10.17487/RFC4861, September 2007, <<http://www.rfc-editor.org/info/rfc4861>>.
- [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>>.
- [RFC6991] Schoenwaelder, J., Ed., "Common YANG Data Types", RFC 6991, DOI 10.17487/RFC6991, July 2013, <<http://www.rfc-editor.org/info/rfc6991>>.
- [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>>.
- [RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <<http://www.rfc-editor.org/info/rfc7950>>.

13.2. Informative References

- [RFC6087] Bierman, A., "Guidelines for Authors and Reviewers of YANG Data Model Documents", RFC 6087, DOI 10.17487/RFC6087, January 2011, <<http://www.rfc-editor.org/info/rfc6087>>.

- [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>>.
- [RFC7895] Bierman, A., Bjorklund, M., and K. Watsen, "YANG Module Library", RFC 7895, DOI 10.17487/RFC7895, June 2016, <<http://www.rfc-editor.org/info/rfc7895>>.
- [RFC7951] Lhotka, L., "JSON Encoding of Data Modeled with YANG", RFC 7951, DOI 10.17487/RFC7951, August 2016, <<http://www.rfc-editor.org/info/rfc7951>>.

Appendix A. The Complete Data Trees

This appendix presents the complete configuration and state data trees of the core routing data model. See Section 2.2 for an explanation of the symbols used. Data type of every leaf node is shown near the right end of the corresponding line.

A.1. Configuration Data

```

+--rw routing
  +--rw router-id?                yang:dotted-quad
  +--rw control-plane-protocols
    +--rw control-plane-protocol* [type name]
      +--rw type                identityref
      +--rw name                 string
      +--rw description?        string
      +--rw static-routes
        +--rw v6ur:ipv6
          +--rw v6ur:route* [destination-prefix]
            +--rw v6ur:destination-prefix  inet:ipv6-prefix
            +--rw v6ur:description?       string
            +--rw v6ur:next-hop
              +--rw (v6ur:next-hop-options)
                +--:(v6ur:simple-next-hop)
                | +--rw v6ur:outgoing-interface?
                | +--rw v6ur:next-hop-address?
                +--:(v6ur:special-next-hop)
                | +--rw v6ur:special-next-hop?  enumeration
                +--:(v6ur:next-hop-list)
                +--rw v6ur:next-hop-list
                  +--rw v6ur:next-hop* [index]
                    +--rw v6ur:index          string
                    +--rw v6ur:outgoing-interface?
                    +--rw v6ur:next-hop-address?
          +--rw v4ur:ipv4
            +--rw v4ur:route* [destination-prefix]
              +--rw v4ur:destination-prefix  inet:ipv4-prefix
              +--rw v4ur:description?       string
              +--rw v4ur:next-hop
                +--rw (v4ur:next-hop-options)
                  +--:(v4ur:simple-next-hop)
                  | +--rw v4ur:outgoing-interface?
                  | +--rw v4ur:next-hop-address?
                  +--:(v4ur:special-next-hop)
                  | +--rw v4ur:special-next-hop?  enumeration
                  +--:(v4ur:next-hop-list)
                  +--rw v4ur:next-hop-list
                    +--rw v4ur:next-hop* [index]
                      +--rw v4ur:index          string
                      +--rw v4ur:outgoing-interface?
                      +--rw v4ur:next-hop-address?
        +--rw ribs
          +--rw rib* [name]
            +--rw name                string
            +--rw address-family?     identityref
            +--rw description?        string

```

A.2. State Data

```

+--ro routing-state
|
|  +--ro router-id?                yang:dotted-quad
|  +--ro interfaces
|  |  +--ro interface*            if:interface-state-ref
|  +--ro control-plane-protocols
|  |  +--ro control-plane-protocol* [type name]
|  |  |  +--ro type                identityref
|  |  |  +--ro name                string
|  +--ro ribs
|  |  +--ro rib* [name]
|  |  |  +--ro name                string
|  |  |  +--ro 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?
|  |  |  |  |  |  |  +--ro v6ur:next-hop-address?
|  |  |  |  |  |  |  +--ro v4ur:next-hop-address?
|  |  |  |  |  |  +--:(special-next-hop)
|  |  |  |  |  |  |  +--ro special-next-hop?          enumeration
|  |  |  |  |  |  +--:(next-hop-list)
|  |  |  |  |  |  |  +--ro next-hop-list
|  |  |  |  |  |  |  |  +--ro next-hop*
|  |  |  |  |  |  |  |  |  +--ro outgoing-interface?
|  |  |  |  |  |  |  |  |  +--ro v6ur:address?
|  |  |  |  |  |  |  |  |  +--ro v4ur:address?
|  |  |  |  +--ro source-protocol    identityref
|  |  |  +--ro active?                empty
|  |  |  +--ro last-updated?         yang:date-and-time
|  |  |  +--ro v6ur:destination-prefix? inet:ipv6-prefix
|  |  |  +--ro v4ur:destination-prefix? inet:ipv4-prefix
|  +---x active-route
|  |  +---w input
|  |  |  +---w v6ur:destination-address?  inet:ipv6-address
|  |  |  +---w v4ur:destination-address?  inet:ipv4-address
|  |  +--ro output
|  |  |  +--ro route
|  |  |  |  +--ro next-hop
|  |  |  |  |  +--ro (next-hop-options)
|  |  |  |  |  |  +--:(simple-next-hop)
|  |  |  |  |  |  |  +--ro outgoing-interface?
|  |  |  |  |  |  |  +--ro v6ur:next-hop-address?

```



```
module example-rip {
  yang-version "1.1";
  namespace "http://example.com/rip";
  prefix "rip";
  import ietf-interfaces {
    prefix "if";
  }
  import ietf-routing {
    prefix "rt";
  }
  identity rip {
    base rt:routing-protocol;
    description
      "Identity for the RIP routing protocol.";
  }
  typedef rip-metric {
    type uint8 {
      range "0..16";
    }
  }
  grouping route-content {
    description
      "This grouping defines RIP-specific route attributes.";
    leaf metric {
      type rip-metric;
    }
    leaf tag {
      type uint16;
      default "0";
      description
        "This leaf may be used to carry additional info, e.g. AS
        number.";
    }
  }
  augment "/rt:routing-state/rt:ribs/rt:rib/rt:routes/rt:route" {
    when "derived-from-or-self(rt:source-protocol, 'rip:rip')" {
      description
        "This augment is only valid for a routes whose source
        protocol is RIP.";
    }
  }
}
```

```
    }
    description
      "RIP-specific route attributes.";
    uses route-content;
  }

  augment "/rt:routing-state/rt:ribs/rt:rib/rt:active-route/"
    + "rt:output/rt:route" {
    description
      "RIP-specific route attributes in the output of 'active-route'
      RPC.";
    uses route-content;
  }

  augment "/rt:routing/rt:control-plane-protocols/"
    + "rt:control-plane-protocol" {
    when "derived-from-or-self(rt:type, 'rip:rip')" {
      description
        "This augment is only valid for a routing protocol instance
        of type 'rip'.";
    }
    container rip {
      presence "RIP configuration";
      description
        "RIP instance configuration.";
      container interfaces {
        description
          "Per-interface RIP configuration.";
        list interface {
          key "name";
          description
            "RIP is enabled on interfaces that have an entry in this
            list, unless 'enabled' is set to 'false' for that
            entry.";
          leaf name {
            type if:interface-ref;
          }
          leaf enabled {
            type boolean;
            default "true";
          }
          leaf metric {
            type rip-metric;
            default "1";
          }
        }
      }
      leaf update-interval {
```

```

    type uint8 {
      range "10..60";
    }
    units "seconds";
    default "30";
    description
      "Time interval between periodic updates.";
  }
}
}
}

```

Appendix D. Data Tree Example

This section contains an example instance data tree in the JSON encoding [RFC7951], containing both configuration and state data. The data conforms to a data model that is defined by the following YANG library specification [RFC7895]:

```

{
  "ietf-yang-library:modules-state": {
    "module-set-id": "c2elf54169aa7f36e1a6e8d0865d441d3600f9c4",
    "module": [
      {
        "name": "ietf-routing",
        "revision": "2016-11-03",
        "feature": [
          "multiple-ribs",
          "router-id"
        ],
        "namespace": "urn:ietf:params:xml:ns:yang:ietf-routing",
        "conformance-type": "implement"
      },
      {
        "name": "ietf-ipv4-unicast-routing",
        "revision": "2016-11-03",
        "namespace":
          "urn:ietf:params:xml:ns:yang:ietf-ipv4-unicast-routing",
        "conformance-type": "implement"
      },
      {
        "name": "ietf-ipv6-unicast-routing",
        "revision": "2016-11-03",
        "namespace":
          "urn:ietf:params:xml:ns:yang:ietf-ipv6-unicast-routing",
        "conformance-type": "implement"
      }
    ]
  }
}

```

```
    "name": "ietf-interfaces",
    "revision": "2014-05-08",
    "namespace": "urn:ietf:params:xml:ns:yang:ietf-interfaces",
    "conformance-type": "implement"
  },
  {
    "name": "ietf-inet-types",
    "namespace": "urn:ietf:params:xml:ns:yang:ietf-inet-types",
    "revision": "2013-07-15",
    "conformance-type": "import"
  },
  {
    "name": "ietf-yang-types",
    "namespace": "urn:ietf:params:xml:ns:yang:ietf-yang-types",
    "revision": "2013-07-15",
    "conformance-type": "import"
  },
  {
    "name": "iana-if-type",
    "namespace": "urn:ietf:params:xml:ns:yang:iana-if-type",
    "revision": "",
    "conformance-type": "implement"
  },
  {
    "name": "ietf-ip",
    "revision": "2014-06-16",
    "namespace": "urn:ietf:params:xml:ns:yang:ietf-ip",
    "conformance-type": "implement"
  }
]
}
```

A simple network set-up as shown in Figure 3 is assumed: router "A" uses static default routes with the "ISP" router as the next-hop. IPv6 router advertisements are configured only on the "eth1" interface and disabled on the upstream "eth0" interface.

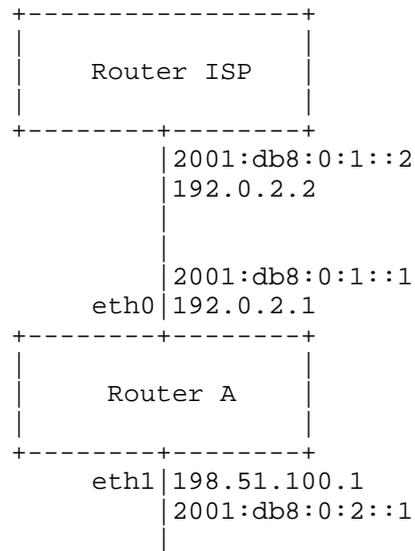


Figure 3: Example network configuration

The instance data tree could then be as follows:

```

{
  "ietf-interfaces:interfaces": {
    "interface": [
      {
        "name": "eth0",
        "type": "iana-if-type:ethernetCsmacd",
        "description": "Uplink to ISP.",
        "ietf-ip:ipv4": {
          "address": [
            {
              "ip": "192.0.2.1",
              "prefix-length": 24
            }
          ]
        },
        "forwarding": true
      },
      {
        "ietf-ip:ipv6": {
          "address": [
            {
              "ip": "2001:0db8:0:1::1",
              "prefix-length": 64
            }
          ]
        },
        "forwarding": true,

```

```
        "autoconf": {
          "create-global-addresses": false
        }
      },
    {
      "name": "eth1",
      "type": "iana-if-type:ethernetCsmacd",
      "description": "Interface to the internal network.",
      "ietf-ip:ipv4": {
        "address": [
          {
            "ip": "198.51.100.1",
            "prefix-length": 24
          }
        ],
        "forwarding": true
      },
      "ietf-ip:ipv6": {
        "address": [
          {
            "ip": "2001:0db8:0:2::1",
            "prefix-length": 64
          }
        ],
        "forwarding": true,
        "autoconf": {
          "create-global-addresses": false
        },
        "ietf-ipv6-unicast-routing:ipv6-router-advertisements": {
          "send-advertisements": true
        }
      }
    }
  ]
},
"ietf-interfaces:interfaces-state": {
  "interface": [
    {
      "name": "eth0",
      "type": "iana-if-type:ethernetCsmacd",
      "phys-address": "00:0C:42:E5:B1:E9",
      "oper-status": "up",
      "statistics": {
        "discontinuity-time": "2015-10-24T17:11:27+02:00"
      },
      "ietf-ip:ipv4": {
        "forwarding": true,

```

```
        "mtu": 1500,
        "address": [
            {
                "ip": "192.0.2.1",
                "prefix-length": 24
            }
        ]
    },
    "ietf-ip:ipv6": {
        "forwarding": true,
        "mtu": 1500,
        "address": [
            {
                "ip": "2001:0db8:0:1::1",
                "prefix-length": 64
            }
        ],
        "ietf-ipv6-unicast-routing:ipv6-router-advertisements": {
            "send-advertisements": true,
            "prefix-list": {
                "prefix": [
                    {
                        "prefix-spec": "2001:db8:0:2::/64"
                    }
                ]
            }
        }
    }
},
{
    "name": "eth1",
    "type": "iana-if-type:ethernetCsmacd",
    "phys-address": "00:0C:42:E5:B1:EA",
    "oper-status": "up",
    "statistics": {
        "discontinuity-time": "2015-10-24T17:11:29+02:00"
    },
    "ietf-ip:ipv4": {
        "forwarding": true,
        "mtu": 1500,
        "address": [
            {
                "ip": "198.51.100.1",
                "prefix-length": 24
            }
        ]
    },
    "ietf-ip:ipv6": {
```

```
    "forwarding": true,
    "mtu": 1500,
    "address": [
      {
        "ip": "2001:0db8:0:2::1",
        "prefix-length": 64
      }
    ],
    "ietf-ipv6-unicast-routing:ipv6-router-advertisements": {
      "send-advertisements": true,
      "prefix-list": {
        "prefix": [
          {
            "prefix-spec": "2001:db8:0:2::/64"
          }
        ]
      }
    }
  }
]
},
"ietf-routing:routing": {
  "router-id": "192.0.2.1",
  "control-plane-protocols": {
    "control-plane-protocol": [
      {
        "type": "ietf-routing:static",
        "name": "st0",
        "description":
          "Static routing is used for the internal network.",
        "static-routes": {
          "ietf-ipv4-unicast-routing:ipv4": {
            "route": [
              {
                "destination-prefix": "0.0.0.0/0",
                "next-hop": {
                  "next-hop-address": "192.0.2.2"
                }
              }
            ]
          }
        },
        "ietf-ipv6-unicast-routing:ipv6": {
          "route": [
            {
              "destination-prefix": "::/0",
              "next-hop": {
                "next-hop-address": "2001:db8:0:1::2"
              }
            }
          ]
        }
      }
    ]
  }
}
```

```

    }
  ]
}
},
"ietf-routing:routing-state": {
  "interfaces": {
    "interface": [
      "eth0",
      "eth1"
    ]
  },
  "control-plane-protocols": {
    "control-plane-protocol": [
      {
        "type": "ietf-routing:static",
        "name": "st0"
      }
    ]
  },
  "ribs": {
    "rib": [
      {
        "name": "ipv4-master",
        "address-family":
          "ietf-ipv4-unicast-routing:ipv4-unicast",
        "default-rib": true,
        "routes": {
          "route": [
            {
              "ietf-ipv4-unicast-routing:destination-prefix":
                "192.0.2.1/24",
              "next-hop": {
                "outgoing-interface": "eth0"
              },
            },
            {
              "route-preference": 0,
              "source-protocol": "ietf-routing:direct",
              "last-updated": "2015-10-24T17:11:27+02:00"
            }
          ],
          {
            "ietf-ipv4-unicast-routing:destination-prefix":
              "198.51.100.0/24",
            "next-hop": {
              "outgoing-interface": "eth1"
            }
          }
        }
      }
    ]
  }
}

```

```

    },
    "source-protocol": "ietf-routing:direct",
    "route-preference": 0,
    "last-updated": "2015-10-24T17:11:27+02:00"
  },
  {
    "ietf-ipv4-unicast-routing:destination-prefix":
      "0.0.0.0/0",
    "source-protocol": "ietf-routing:static",
    "route-preference": 5,
    "next-hop": {
      "ietf-ipv4-unicast-routing:next-hop-address":
        "192.0.2.2"
    },
    "last-updated": "2015-10-24T18:02:45+02:00"
  }
]
}
},
{
  "name": "ipv6-master",
  "address-family":
    "ietf-ipv6-unicast-routing:ipv6-unicast",
  "default-rib": true,
  "routes": {
    "route": [
      {
        "ietf-ipv6-unicast-routing:destination-prefix":
          "2001:db8:0:1::/64",
        "next-hop": {
          "outgoing-interface": "eth0"
        },
        "source-protocol": "ietf-routing:direct",
        "route-preference": 0,
        "last-updated": "2015-10-24T17:11:27+02:00"
      },
      {
        "ietf-ipv6-unicast-routing:destination-prefix":
          "2001:db8:0:2::/64",
        "next-hop": {
          "outgoing-interface": "eth1"
        },
        "source-protocol": "ietf-routing:direct",
        "route-preference": 0,
        "last-updated": "2015-10-24T17:11:27+02:00"
      },
      {
        "ietf-ipv6-unicast-routing:destination-prefix":

```


E.4. Changes Between Versions -21 and -22

- o Added "next-hop-list" as a new case of the "next-hop-options" choice.
- o Renamed "routing protocol" to "control plane protocol" in both the YANG modules and I-D text.

E.5. Changes Between Versions -20 and -21

- o Routing instances were removed.
- o IPv6 RA parameters were moved to the "ietf-ipv6-router-advertisements".

E.6. Changes Between Versions -19 and -20

- o Assignment of L3 interfaces to routing instances is now part of interface configuration.
- o Next-hop options in configuration were aligned with state data.
- o It is recommended to enclose protocol-specific configuration in a presence container.

E.7. Changes Between Versions -18 and -19

- o The leaf "route-preference" was removed from the "routing-protocol" container in both "routing" and "routing-state".
- o The "vrf-routing-instance" identity was added in support of a common routing-instance type in addition to the "default-routing-instance".
- o Removed "enabled" switch from "routing-protocol".

E.8. Changes Between Versions -17 and -18

- o The container "ribs" was moved under "routing-instance" (in both "routing" and "routing-state").
- o Typedefs "rib-ref" and "rib-state-ref" were removed.
- o Removed "recipient-ribs" (both state and configuration).
- o Removed "connected-ribs" from "routing-protocol" (both state and configuration).

- o Configuration and state data for IPv6 RA were moved under "if:interface" and "if:interface-state".
- o Assignment of interfaces to routing instances now use leaf-list rather than list (both config and state). The opposite reference from "if:interface" to "rt:routing-instance" was changed to a single leaf (an interface cannot belong to multiple routing instances).
- o Specification of a default RIB is now a simple flag under "rib" (both config and state).
- o Default RIBs are marked by a flag in state data.

E.9. Changes Between Versions -16 and -17

- o Added Acee as a co-author.
- o Removed all traces of route filters.
- o Removed numeric IDs of list entries in state data.
- o Removed all next-hop cases except "simple-next-hop" and "special-next-hop".
- o Removed feature "multipath-routes".
- o Augmented "ietf-interfaces" module with a leaf-list of leafrefs pointing from state data of an interface entry to the routing instance(s) to which the interface is assigned.

E.10. Changes Between Versions -15 and -16

- o Added 'type' as the second key component of 'routing-protocol', both in configuration and state data.
- o The restriction of no more than one connected RIB per address family was removed.
- o Removed the 'id' key of routes in RIBs. This list has no keys anymore.
- o Remove the 'id' key from static routes and make 'destination-prefix' the only key.
- o Added 'route-preference' as a new attribute of routes in RIB.
- o Added 'active' as a new attribute of routes in RIBs.

- o Renamed RPC operation 'active-route' to 'fib-route'.
- o Added 'route-preference' as a new parameter of routing protocol instances, both in configuration and state data.
- o Renamed identity 'rt:standard-routing-instance' to 'rt:default-routing-instance'.
- o Added next-hop lists to state data.
- o Added two cases for specifying next-hops indirectly - via a new RIB or a recursive list of next-hops.
- o Reorganized next-hop in static routes.
- o Removed all 'if-feature' statements from state data.

E.11. Changes Between Versions -14 and -15

- o Removed all defaults from state data.
- o Removed default from 'cur-hop-limit' in config.

E.12. Changes Between Versions -13 and -14

- o Removed dependency of 'connected-ribs' on the 'multiple-ribs' feature.
- o Removed default value of 'cur-hop-limit' in state data.
- o Moved parts of descriptions and all references on IPv6 RA parameters from state data to configuration.
- o Added reference to RFC 6536 in the Security section.

E.13. Changes Between Versions -12 and -13

- o Wrote appendix about minimum implementation.
- o Remove "when" statement for IPv6 router interface state data - it was dependent on a config value that may not be present.
- o Extra container for the next-hop list.
- o Names rather than numeric ids are used for referring to list entries in state data.

- o Numeric ids are always declared as mandatory and unique. Their description states that they are ephemeral.
- o Descriptions of "name" keys in state data lists are required to be persistent.
- o
- o Removed "if-feature multiple-ribs;" from connected-ribs.
- o "rib-name" instead of "name" is used as the name of leafref nodes.
- o "next-hop" instead of "nexthop" or "gateway" used throughout, both in node names and text.

E.14. Changes Between Versions -11 and -12

- o Removed feature "advanced-router" and introduced two features instead: "multiple-ribs" and "multipath-routes".
- o Unified the keys of config and state versions of "routing-instance" and "rib" lists.
- o Numerical identifiers of state list entries are not keys anymore, but they are constrained using the "unique" statement.
- o Updated acknowledgements.

E.15. Changes Between Versions -10 and -11

- o Migrated address families from IANA enumerations to identities.
- o Terminology and node names aligned with the I2RS RIB model: router -> routing instance, routing table -> RIB.
- o Introduced uint64 keys for state lists: routing-instance, rib, route, nexthop.
- o Described the relationship between system-controlled and user-controlled list entries.
- o Feature "user-defined-routing-tables" changed into "advanced-router".
- o Made nexthop into a choice in order to allow for nexthop-list (I2RS requirement).

- o Added nexthop-list with entries having priorities (backup) and weights (load balancing).
- o Updated bibliography references.

E.16. Changes Between Versions -09 and -10

- o Added subtree for state data ("/routing-state").
- o Terms "system-controlled entry" and "user-controlled entry" defined and used.
- o New feature "user-defined-routing-tables". Nodes that are useful only with user-defined routing tables are now conditional.
- o Added grouping "router-id".
- o In routing tables, "source-protocol" attribute of routes now reports only protocol type, and its datatype is "identityref".
- o Renamed "main-routing-table" to "default-routing-table".

E.17. Changes Between Versions -08 and -09

- o Fixed "must" expression for "connected-routing-table".
- o Simplified "must" expression for "main-routing-table".
- o Moved per-interface configuration of a new routing protocol under 'routing-protocol'. This also affects the 'example-rip' module.

E.18. Changes Between Versions -07 and -08

- o Changed reference from RFC6021 to RFC6021bis.

E.19. Changes Between Versions -06 and -07

- o The contents of <get-reply> in Appendix D was updated: "eth[01]" is used as the value of "location", and "forwarding" is on for both interfaces and both IPv4 and IPv6.
- o The "must" expression for "main-routing-table" was modified to avoid redundant error messages reporting address family mismatch when "name" points to a non-existent routing table.
- o The default behavior for IPv6 RA prefix advertisements was clarified.

- o Changed type of "rt:router-id" to "ip:dotted-quad".
- o Type of "rt:router-id" changed to "yang:dotted-quad".
- o Fixed missing prefixes in XPath expressions.

E.20. Changes Between Versions -05 and -06

- o Document title changed: "Configuration" was replaced by "Management".
- o New typedefs "routing-table-ref" and "route-filter-ref".
- o Double slashes "//" were removed from XPath expressions and replaced with the single "/".
- o Removed uniqueness requirement for "router-id".
- o Complete data tree is now in Appendix A.
- o Changed type of "source-protocol" from "leafref" to "string".
- o Clarified the relationship between routing protocol instances and connected routing tables.
- o Added a must constraint saying that a routing table connected to the direct pseudo-protocol must not be a main routing table.

E.21. Changes Between Versions -04 and -05

- o Routing tables are now global, i.e., "routing-tables" is a child of "routing" rather than "router".
- o "must" statement for "static-routes" changed to "when".
- o Added "main-routing-tables" containing references to main routing tables for each address family.
- o Removed the defaults for "address-family" and "safi" and made them mandatory.
- o Removed the default for route-filter/type and made this leaf mandatory.
- o If there is no active route for a given destination, the "active-route" RPC returns no output.
- o Added "enabled" switch under "routing-protocol".

- o Added "router-type" identity and "type" leaf under "router".
- o Route attribute "age" changed to "last-updated", its type is "yang:date-and-time".
- o The "direct" pseudo-protocol is always connected to main routing tables.
- o Entries in the list of connected routing tables renamed from "routing-table" to "connected-routing-table".
- o Added "must" constraint saying that a routing table must not be its own recipient.

E.22. Changes Between Versions -03 and -04

- o Changed "error-tag" for both RPC operations from "missing element" to "data-missing".
- o Removed the decrementing behavior for advertised IPv6 prefix parameters "valid-lifetime" and "preferred-lifetime".
- o Changed the key of the static route lists from "seqno" to "id" because the routes needn't be sorted.
- o Added 'must' constraint saying that "preferred-lifetime" must not be greater than "valid-lifetime".

E.23. Changes Between Versions -02 and -03

- o Module "iana-afn-safi" moved to I-D "iana-if-type".
- o Removed forwarding table.
- o RPC "get-route" changed to "active-route". Its output is a list of routes (for multi-path routing).
- o New RPC "route-count".
- o For both RPCs, specification of negative responses was added.
- o Relaxed separation of router instances.
- o Assignment of interfaces to router instances needn't be disjoint.
- o Route filters are now global.
- o Added "allow-all-route-filter" for symmetry.

- o Added Section 6 about interactions with "ietf-interfaces" and "ietf-ip".
- o Added "router-id" leaf.
- o Specified the names for IPv4/IPv6 unicast main routing tables.
- o Route parameter "last-modified" changed to "age".
- o Added container "recipient-routing-tables".

E.24. Changes Between Versions -01 and -02

- o Added module "ietf-ipv6-unicast-routing".
- o The example in Appendix D now uses IP addresses from blocks reserved for documentation.
- o Direct routes appear by default in the forwarding table.
- o Network layer interfaces must be assigned to a router instance. Additional interface configuration may be present.
- o The "when" statement is only used with "augment", "must" is used elsewhere.
- o Additional "must" statements were added.
- o The "route-content" grouping for IPv4 and IPv6 unicast now includes the material from the "ietf-routing" version via "uses rt:route-content".
- o Explanation of symbols in the tree representation of data model hierarchy.

E.25. Changes Between Versions -00 and -01

- o AFN/SAFI-independent stuff was moved to the "ietf-routing" module.
- o Typedefs for AFN and SAFI were placed in a separate "iana-afn-safi" module.
- o Names of some data nodes were changed, in particular "routing-process" is now "router".
- o The restriction of a single AFN/SAFI per router was lifted.
- o RPC operation "delete-route" was removed.

- o Illegal XPath references from "get-route" to the datastore were fixed.
- o Section "Security Considerations" was written.

Authors' Addresses

Ladislav Lhotka
CZ.NIC

Email: lhotka@nic.cz

Acee Linden
Cisco Systems

Email: acee@cisco.com

Routing Area Working Group
Internet-Draft
Intended status: Standards Track
Expires: August 19, 2016

G. Enyedi
A. Csaszar
Ericsson
A. Atlas
C. Bowers
Juniper Networks
A. Gopalan
University of Arizona
February 16, 2016

An Algorithm for Computing Maximally Redundant Trees for IP/LDP Fast-
Reroute
draft-ietf-rtgwg-mrt-frr-algorithm-09

Abstract

A solution for IP and LDP Fast-Reroute using Maximally Redundant Trees is presented in draft-ietf-rtgwg-mrt-frr-architecture. This document defines the associated MRT Lowpoint algorithm that is used in the Default MRT profile to compute both the necessary Maximally Redundant Trees with their associated next-hops and the alternates to select for MRT-FRR.

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 August 19, 2016.

Copyright Notice

Copyright (c) 2016 IETF Trust and the persons identified as the document authors. All rights reserved.

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

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

Table of Contents

1.	Introduction	3
2.	Requirements Language	5
3.	Terminology and Definitions	5
4.	Algorithm Key Concepts	6
4.1.	Partial Ordering for Disjoint Paths	7
4.2.	Finding an Ear and the Correct Direction	8
4.3.	Low-Point Values and Their Uses	11
4.4.	Blocks in a Graph	14
4.5.	Determining Local-Root and Assigning Block-ID	16
5.	MRT Lowpoint Algorithm Specification	18
5.1.	Interface Ordering	18
5.2.	MRT Island Identification	21
5.3.	GADAG Root Selection	21
5.4.	Initialization	22
5.5.	Constructing the GADAG using lowpoint inheritance	23
5.6.	Augmenting the GADAG by directing all links	25
5.7.	Compute MRT next-hops	29
5.7.1.	MRT next-hops to all nodes ordered with respect to the computing node	29
5.7.2.	MRT next-hops to all nodes not ordered with respect to the computing node	30
5.7.3.	Computing Redundant Tree next-hops in a 2-connected Graph	31
5.7.4.	Generalizing for a graph that isn't 2-connected	33
5.7.5.	Complete Algorithm to Compute MRT Next-Hops	34
5.8.	Identify MRT alternates	36
5.9.	Named Proxy-Nodes	43
5.9.1.	Determining Proxy-Node Attachment Routers	43
5.9.2.	Computing if an Island Neighbor (IN) is loop-free	44
5.9.3.	Computing MRT Next-Hops for Proxy-Nodes	46
5.9.4.	Computing MRT Alternates for Proxy-Nodes	52
6.	MRT Lowpoint Algorithm: Next-hop conformance	60
7.	Broadcast interfaces	60
7.1.	Computing MRT next-hops on broadcast networks	61
7.2.	Using MRT next-hops as alternates in the event of failures on broadcast networks	62
8.	Evaluation of Alternative Methods for Constructing GADAGs	63
9.	Implementation Status	64

10. Operational Considerations	65
10.1. GADAG Root Selection	65
10.2. Destination-rooted GADAGs	65
11. Acknowledgements	66
12. IANA Considerations	66
13. Security Considerations	66
14. References	66
14.1. Normative References	66
14.2. Informative References	66
Appendix A. Python Implementation of MRT Lowpoint Algorithm . .	67
Appendix B. Constructing a GADAG using SPFs	108
Appendix C. Constructing a GADAG using a hybrid method	113
Authors' Addresses	115

1. Introduction

MRT Fast-Reroute requires that packets can 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 which experiences a local failure must also have pre-determined which alternate to use. This document defines how to compute these three things for use in MRT-FRR and describes the algorithm design decisions and rationale. The algorithm is based on those presented in [MRTLinear] and expanded in [EnyediThesis]. The MRT Lowpoint algorithm is required for implementation when the Default MRT profile is implemented.

Just as packets routed on a hop-by-hop basis require that each router compute a shortest-path tree which is consistent, it is necessary for each router to compute the MRT-Blue next-hops and MRT-Red next-hops in a consistent fashion. This document defines the MRT Lowpoint algorithm to be used as a standard in the default MRT profile for MRT-FRR.

As now, a router's FIB will contain primary next-hops for the current shortest-path tree for forwarding traffic. In addition, a router's FIB will contain primary next-hops for the MRT-Blue for forwarding received traffic on the MRT-Blue and primary next-hops for the MRT-Red for forwarding received traffic on the MRT-Red.

What alternate next-hops a point-of-local-repair (PLR) selects need not be consistent - but loops must be prevented. To reduce congestion, it is possible for multiple alternate next-hops to be selected; in the context of MRT alternates, each of those alternate next-hops would be equal-cost paths.

This document defines an algorithm for selecting an appropriate MRT alternate for consideration. Other alternates, e.g. LFAs that are

downstream paths, may be preferred when available. See the Operational Considerations section of [I-D.ietf-rtgwg-mrt-frr-architecture] for a more detailed discussion of combining MRT alternates with those produced by other FRR technologies.

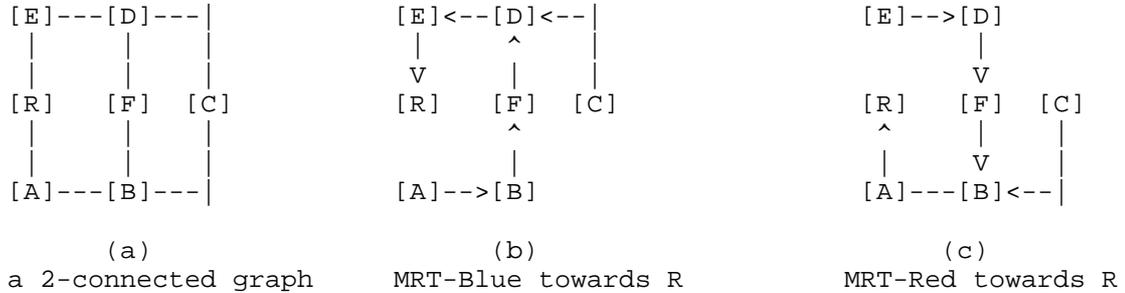
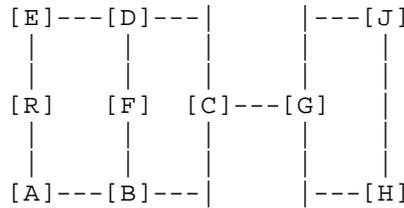
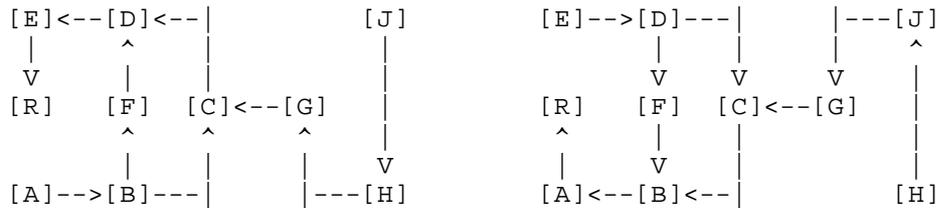


Figure 1

The MRT Lowpoint algorithm can handle arbitrary network topologies where the whole network graph is not 2-connected, as in Figure 2, as well as the easier case where the network graph is 2-connected (Figure 1). Each MRT is a spanning tree. The pair of MRTs provide two paths from every node X to the root of the MRTs. Those paths 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.



(a) a graph that isn't 2-connected



(b) MRT-Blue towards R

(c) MRT-Red towards R

Figure 2

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 and Definitions

Please see the Terminology section of [I-D.ietf-rtgwg-mrt-frr-architecture] for a complete list of terminology relevant to this draft. The list below does not repeat terminology introduced in that draft.

spanning tree: A tree containing links that connects all nodes in the network graph.

back-edge: In the context of a spanning tree computed via a depth-first search, a back-edge is a link that connects a descendant of a node x with an ancestor of x.

partial ADAG: A subset of an ADAG that doesn't yet contain all the nodes in the block. A partial ADAG is created during the MRT algorithm and then expanded until all nodes in the block are included and it is an ADAG.

DFS: Depth-First Search

DFS ancestor: A node n is a DFS ancestor of x if n is on the DFS-tree path from the DFS root to x .

DFS descendant: A node n is a DFS descendant of x if x is on the DFS-tree path from the DFS root to n .

ear: A path along not-yet-included-in-the-GADAG nodes that starts at a node that is already-included-in-the-GADAG and that ends at a node that is already-included-in-the-GADAG. The starting and ending nodes may be the same node if it is a cut-vertex.

$X \gg Y$ or $Y \ll X$: Indicates the relationship between X and Y in a partial order, such as found in a GADAG. $X \gg Y$ means that X is higher in the partial order than Y . $Y \ll X$ means that Y is lower in the partial order than X .

$X > Y$ or $Y < X$: Indicates the relationship between X and Y in the total order, such as found via a topological sort. $X > Y$ means that X is higher in the total order than Y . $Y < X$ means that Y is lower in the total order than X .

$X ?? Y$: Indicates that X is unordered with respect to Y in the partial order.

UNDIRECTED: In the GADAG, each link is marked as OUTGOING, INCOMING or both. Until the directionality of the link is determined, the link is marked as UNDIRECTED to indicate that its direction hasn't been determined.

OUTGOING: A link marked as OUTGOING has direction in the GADAG from the interface's router to the remote end.

INCOMING: A link marked as INCOMING has direction in the GADAG from the remote end to the interface's router.

4. Algorithm Key Concepts

There are five key concepts that are critical for understanding the MRT Lowpoint algorithm. The first is the idea of partially ordering the nodes in a network graph with regard to each other and to the GADAG root. The second is the idea of finding an ear of nodes and adding them in the correct direction. The third is the idea of a Low-Point value and how it can be used to identify cut-vertices and to find a second path towards the root. The fourth is the idea that a non-2-connected graph is made up of blocks, where a block is a 2-connected cluster, a cut-link or an isolated node. The fifth is the idea of a local-root for each node; this is used to compute ADAGs in each block.

4.1. Partial Ordering for Disjoint Paths

Given any two nodes X and Y in a graph, a particular total order means that either $X < Y$ or $X > Y$ in that total order. An example would be a graph where the nodes are ranked based upon their unique IP loopback addresses. In a partial order, there may be some nodes for which it can't be determined whether $X << Y$ or $X >> Y$. A partial order can be captured in a directed graph, as shown in Figure 3. In a graphical representation, a link directed from X to Y indicates that X is a neighbor of Y in the network graph and $X << Y$.

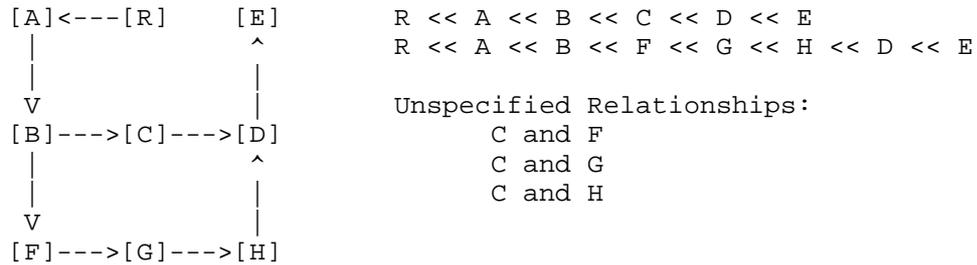


Figure 3: Directed Graph showing a Partial Order

To compute MRTs, the root of the MRTs is at both the very bottom and the very top of the partial ordering. This means that from any node X, one can pick nodes higher in the order until the root is reached. Similarly, from any node X, one can pick nodes lower in the order until the root is reached. For instance, in Figure 4, from G the higher nodes picked can be traced by following the directed links and are H, D, E and R. Similarly, from G the lower nodes picked can be traced by reversing the directed links and are F, B, A, and R. A graph that represents this modified partial order is no longer a DAG; it is termed an Almost DAG (ADAG) because if the links directed to the root were removed, it would be a DAG.

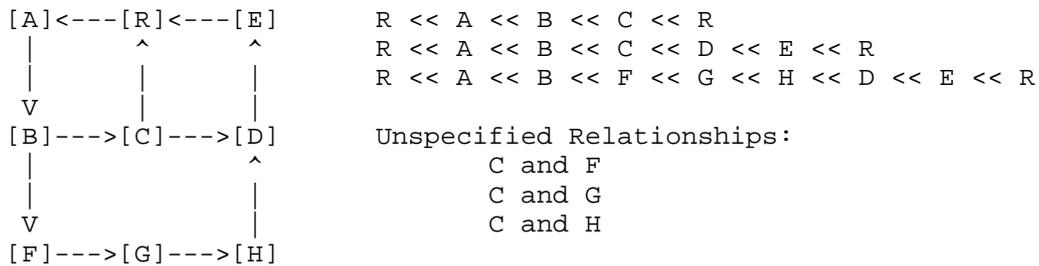


Figure 4: ADAG showing a Partial Order with R lowest and highest

Most importantly, if a node $Y \gg X$, then Y can only appear on the increasing path from X to the root and never on the decreasing path. Similarly, if a node $Z \ll X$, then Z can only appear on the decreasing path from X to the root and never on the increasing path.

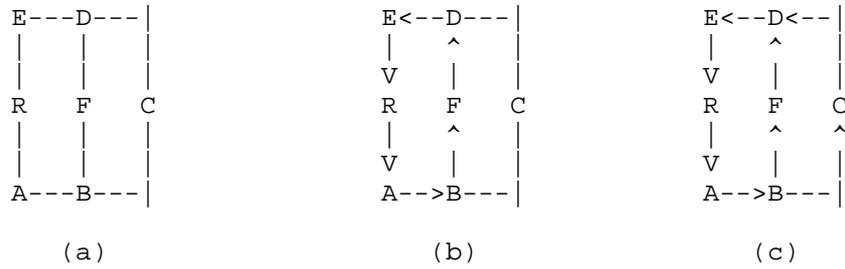
When following the increasing paths, it is possible to pick multiple higher nodes and still have the certainty that those paths will be disjoint from the decreasing paths. E.g. in the previous example node B has multiple possibilities to forward packets along an increasing path: it can either forward packets to C or F .

4.2. Finding an Ear and the Correct Direction

For simplicity, the basic idea of creating a GADAG by adding ears is described assuming that the network graph is a single 2-connected cluster so that an ADAG is sufficient. Generalizing to multiple blocks is done by considering the block-roots instead of the GADAG root - and the actual algorithm is given in Section 5.5.

In order to understand the basic idea of finding an ADAG, first suppose that we have already a partial ADAG, which doesn't contain all the nodes in the block yet, and we want to extend it to cover all the nodes. Suppose that we find a path from a node X to Y such that X and Y are already contained by our partial ADAG, but all the remaining nodes along the path are not added to the ADAG yet. We refer to such a path as an ear.

Recall that our ADAG is closely related to a partial order. More precisely, if we remove root R , the remaining DAG describes a partial order of the nodes. If we suppose that neither X nor Y is the root, we may be able to compare them. If one of them is definitely lesser with respect to our partial order (say $X \ll Y$), we can add the new path to the ADAG in a direction from X to Y . As an example consider Figure 5.



(a) A 2-connected graph
 (b) Partial ADAG (C is not included)
 (c) Resulting ADAG after adding path (or ear) B-C-D

Figure 5

In this partial ADAG, node C is not yet included. However, we can find path B-C-D, where both endpoints are contained by this partial ADAG (we say those nodes are "ready" in the following text), and the remaining node (node C) is not contained yet. If we remove R, the remaining DAG defines a partial order, and with respect to this partial order we can say that $B \ll D$, so we can add the path to the ADAG in the direction from B to D (arcs B->C and C->D are added). If $B \gg D$, we would add the same path in reverse direction.

If in the partial order where an ear's two ends are X and Y, $X \ll Y$, then there must already be a directed path from X to Y in the ADAG. The ear must be added in a direction such that it doesn't create a cycle; therefore the ear must go from X to Y.

In the case, when X and Y are not ordered with each other, we can select either direction for the ear. We have no restriction since neither of the directions can result in a cycle. In the corner case when one of the endpoints of an ear, say X, is the root (recall that the two endpoints must be different), we could use both directions again for the ear because the root can be considered both as smaller and as greater than Y. However, we strictly pick that direction in which the root is lower than Y. The logic for this decision is explained in Section 5.7

A partial ADAG is started by finding a cycle from the root R back to itself. This can be done by selecting a non-ready neighbor N of R and then finding a path from N to R that doesn't use any links between R and N. The direction of the cycle can be assigned either way since it is starting the ordering.

Once a partial ADAG is already present, it will always have a node that is not the root R in it. As a brief proof that a partial ADAG

can always have ears added to it: just select a non-ready neighbor N of a ready node Q , such that Q is not the root R , find a path from N to the root R in the graph with Q removed. This path is an ear where the first node of the ear is Q , the next is N , then the path until the first ready node the path reached (that ready node is the other endpoint of the path). Since the graph is 2-connected, there must be a path from N to R without Q .

It is always possible to select a non-ready neighbor N of a ready node Q so that Q is not the root R . Because the network is 2-connected, N must be connected to two different nodes and only one can be R . Because the initial cycle has already been added to the ADAG, there are ready nodes that are not R . Since the graph is 2-connected, while there are non-ready nodes, there must be a non-ready neighbor N of a ready node that is not R .

```
Generic_Find_Ears_ADAG(root)
  Create an empty ADAG. Add root to the ADAG.
  Mark root as IN_GADAG.
  Select an arbitrary cycle containing root.
  Add the arbitrary cycle to the ADAG.
  Mark cycle's nodes as IN_GADAG.
  Add cycle's non-root nodes to process_list.
  while there exists connected nodes in graph that are not IN_GADAG
    Select a new ear. Let its endpoints be X and Y.
    if Y is root or (Y << X)
      add the ear towards X to the ADAG
    else // (a) X is root or (b) X << Y or (c) X, Y not ordered
      Add the ear towards Y to the ADAG
```

Figure 6: Generic Algorithm to find ears and their direction in 2-connected graph

The algorithm in Figure 6 merely requires that a cycle or ear be selected without specifying how. Regardless of the method for selecting the path, we will get an ADAG. The method used for finding and selecting the ears is important; shorter ears result in shorter paths along the MRTs. The MRT Lowpoint algorithm uses the Low-Point Inheritance method for constructing an ADAG (and ultimately a GADAG). This method is defined in Section 5.5. Other methods for constructing GADAGs are described in Appendix B and Appendix C. An evaluation of these different methods is given in Section 8

As an example, consider Figure 5 again. First, we select the shortest cycle containing R , which can be $R-A-B-F-D-E$ (uniform link costs were assumed), so we get to the situation depicted in Figure 5 (b). Finally, we find a node next to a ready node; that must be node C and assume we reached it from ready node B . We search a path from

C to R without B in the original graph. The first ready node along this is node D, so the open ear is B-C-D. Since $B \ll D$, we add arc B->C and C->D to the ADAG. Since all the nodes are ready, we stop at this point.

4.3. Low-Point Values and Their Uses

A basic way of computing a spanning tree on a network graph is to run a depth-first-search, such as given in Figure 7. This tree has the important property that if there is a link (x, n) , then either n is a DFS ancestor of x or n is a DFS descendant of x . In other words, either n is on the path from the root to x or x is on the path from the root to n .

```

global_variable: dfs_number

DFS_Visit(node x, node parent)
  D(x) = dfs_number
  dfs_number += 1
  x.dfs_parent = parent
  for each link (x, w)
    if D(w) is not set
      DFS_Visit(w, x)

Run_DFS(node gadag_root)
  dfs_number = 0
  DFS_Visit(gadag_root, NONE)

```

Figure 7: Basic Depth-First Search algorithm

Given a node x , one can compute the minimal DFS number of the neighbours of x , i.e. $\min(D(w) \text{ if } (x,w) \text{ is a link})$. This gives the earliest attachment point neighbouring x . What is interesting, though, is what is the earliest attachment point from x and x 's descendants. This is what is determined by computing the Low-Point value.

In order to compute the low point value, the network is traversed using DFS and the vertices are numbered based on the DFS walk. Let this number be represented as $DFS(x)$. All the edges that lead to already visited nodes during DFS walk are back-edges. The back-edges are important because they give information about reachability of a node via another path.

The low point number is calculated by finding:

$$\text{Low}(x) = \text{Minimum of } (\text{DFS}(x), \text{Lowest DFS}(n, x \rightarrow n \text{ is a back-edge}),$$

Lowest Low(n , $x \rightarrow n$ is tree edge in DFS walk)).

A detailed algorithm for computing the low-point value is given in Figure 8. Figure 9 illustrates how the lowpoint algorithm applies to a example graph.

```

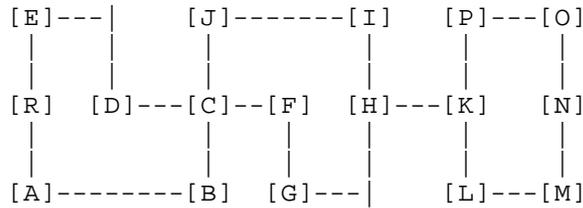
global_variable: dfs_number

Lowpoint_Visit(node x, node parent, interface p_to_x)
  D(x) = dfs_number
  L(x) = D(x)
  dfs_number += 1
  x.dfs_parent = parent
  x.dfs_parent_intf = p_to_x.remote_intf
  x.lowpoint_parent = NONE
  for each ordered_interface intf of x
    if D(intf.remote_node) is not set
      Lowpoint_Visit(intf.remote_node, x, intf)
      if L(intf.remote_node) < L(x)
        L(x) = L(intf.remote_node)
        x.lowpoint_parent = intf.remote_node
        x.lowpoint_parent_intf = intf
    else if intf.remote_node is not parent
      if D(intf.remote_node) < L(x)
        L(x) = D(intf.remote_node)
        x.lowpoint_parent = intf.remote_node
        x.lowpoint_parent_intf = intf

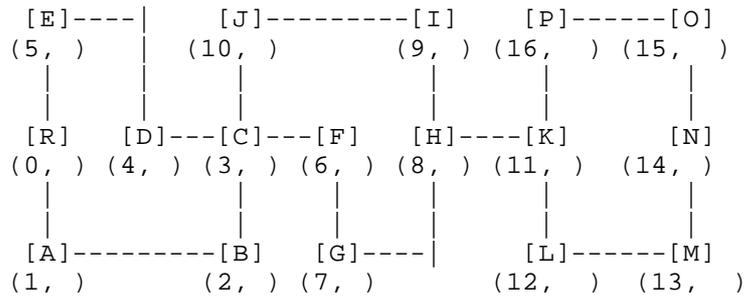
Run_Lowpoint(node gadag_root)
  dfs_number = 0
  Lowpoint_Visit(gadag_root, NONE, NONE)

```

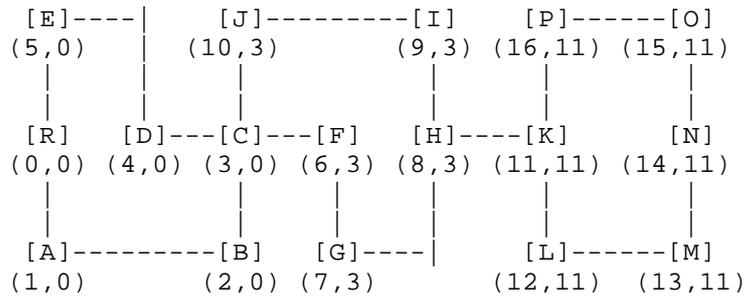
Figure 8: Computing Low-Point value



(a) a non-2-connected graph



(b) with DFS values assigned (D(x), L(x))



(c) with low-point values assigned (D(x), L(x))

Figure 9: Example lowpoint value computation

From the low-point value and lowpoint parent, there are three very useful things which motivate our computation.

First, if there is a child c of x such that $L(c) \geq D(x)$, then there are no paths in the network graph that go from c or its descendants to an ancestor of x - and therefore x is a cut-vertex. In Figure 9, this can be seen by looking at the DFS children of C . C has two children - D and F and $L(F) = 3 = D(C)$ so it is clear that C is a cut-vertex and F is in a block where C is the block's root. $L(D) = 0$

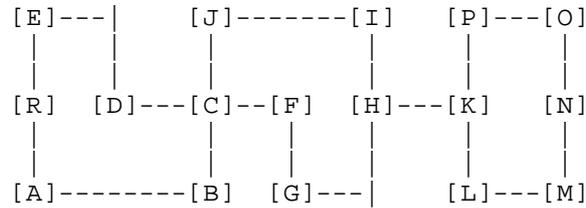
$L(x) < 3 = D(C)$ so D has a path to the ancestors of C; in this case, D can go via E to reach R. Comparing the low-point values of all a node's DFS-children with the node's DFS-value is very useful because it allows identification of the cut-vertices and thus the blocks.

Second, by repeatedly following the path given by `lowpoint_parent`, there is a path from x back to an ancestor of x that does not use the link `[x, x.dfs_parent]` in either direction. The full path need not be taken, but this gives a way of finding an initial cycle and then ears.

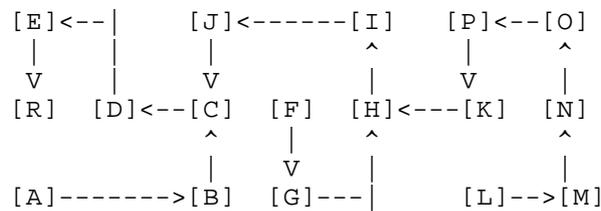
Third, as seen in Figure 9, even if $L(x) < D(x)$, there may be a block that contains both the root and a DFS-child of a node while other DFS-children might be in different blocks. In this example, C's child D is in the same block as R while F is not. It is important to realize that the root of a block may also be the root of another block.

4.4. Blocks in a Graph

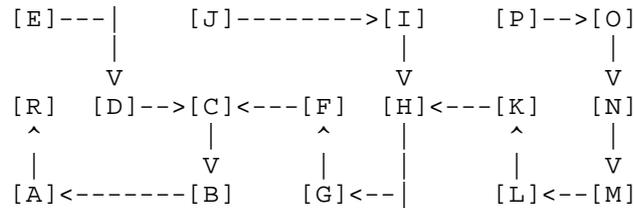
A key idea for the MRT Lowpoint algorithm is that any non-2-connected graph is made up by blocks (e.g. 2-connected clusters, cut-links, and/or isolated nodes). To compute GADAGs and thus MRTs, computation is done in each block to compute ADAGs or Redundant Trees and then those ADAGs or Redundant Trees are combined into a GADAG or MRT.



(a) A graph with four blocks that are:
 three 2-connected clusters
 and one cut-link



(b) MRT-Blue for destination R



(c) MRT-Red for destination R

Figure 10

Consider the example depicted in Figure 10 (a). In this figure, a special graph is presented, showing us all the ways 2-connected clusters can be connected. It has four blocks: block 1 contains R, A, B, C, D, E, block 2 contains C, F, G, H, I, J, block 3 contains K, L, M, N, O, P, and block 4 is a cut-link containing H and K. As can be observed, the first two blocks have one common node (node C) and blocks 2 and 3 do not have any common node, but they are connected through a cut-link that is block 4. No two blocks can have more than one common node, since two blocks with at least two common nodes would qualify as a single 2-connected cluster.

Moreover, observe that if we want to get from one block to another, we must use a cut-vertex (the cut-vertices in this graph are C, H, K), regardless of the path selected, so we can say that all the paths from block 3 along the MRTs rooted at R will cross K first. This observation means that if we want to find a pair of MRTs rooted at R, then we need to build up a pair of RTs in block 3 with K as a root. Similarly, we need to find another pair of RTs in block 2 with C as a root, and finally, we need the last pair of RTs in block 1 with R as a root. When all the trees are selected, we can simply combine them; when a block is a cut-link (as in block 4), that cut-link is added in the same direction to both of the trees. The resulting trees are depicted in Figure 10 (b) and (c).

Similarly, to create a GADAG it is sufficient to compute ADAGs in each block and connect them.

It is necessary, therefore, to identify the cut-vertices, the blocks and identify the appropriate local-root to use for each block.

4.5. Determining Local-Root and Assigning Block-ID

Each node in a network graph has a local-root, which is the cut-vertex (or root) in the same block that is closest to the root. The local-root is used to determine whether two nodes share a common block.

```

Compute_Localroot(node x, node localroot)
  x.localroot = localroot
  for each DFS child node c of x
    if L(c) < D(x) //x is not a cut-vertex
      Compute_Localroot(c, x.localroot)
    else
      mark x as cut-vertex
      Compute_Localroot(c, x)

Compute_Localroot(gadag_root, gadag_root)

```

Figure 11: A method for computing local-roots

There are two different ways of computing the local-root for each node. The stand-alone method is given in Figure 11 and better illustrates the concept; it is used by the GADAG construction methods given in Appendix B and Appendix C. The MRT Lowpoint algorithm computes the local-root for a block as part of computing the GADAG using lowpoint inheritance; the essence of this computation is given in Figure 12. Both methods for computing the local-root produce the same results.

```

Get the current node, s.
Compute an ear(either through lowpoint inheritance
or by following dfs parents) from s to a ready node e.
(Thus, s is not e, if there is such ear.)
if s is e
  for each node x in the ear that is not s
    x.localroot = s
else
  for each node x in the ear that is not s or e
    x.localroot = e.localroot

```

Figure 12: Ear-based method for computing local-roots

Once the local-roots are known, two nodes X and Y are in a common block if and only if one of the following three conditions apply.

- o Y's local-root is X's local-root : They are in the same block and neither is the cut-vertex closest to the root.
- o Y's local-root is X: X is the cut-vertex closest to the root for Y's block
- o Y is X's local-root: Y is the cut-vertex closest to the root for X's block

Once we have computed the local-root for each node in the network graph, we can assign for each node, a block id that represents the block in which the node is present. This computation is shown in Figure 13.

```

global_var: max_block_id

Assign_Block_ID(x, cur_block_id)
  x.block_id = cur_block_id
  foreach DFS child c of x
    if (c.local_root is x)
      max_block_id += 1
      Assign_Block_ID(c, max_block_id)
    else
      Assign_Block_ID(c, cur_block_id)

max_block_id = 0
Assign_Block_ID(gadag_root, max_block_id)

```

Figure 13: Assigning block id to identify blocks

5. MRT Lowpoint Algorithm Specification

The MRT Lowpoint algorithm computes one GADAG that is then used by a router to determine its MRT-Blue and MRT-Red next-hops to all destinations. Finally, based upon that information, alternates are selected for each next-hop to each destination. The different parts of this algorithm are described below.

- o Order the interfaces in the network graph. [See Section 5.1]
- o Compute the local MRT Island for the particular MRT Profile. [See Section 5.2]
- o Select the root to use for the GADAG. [See Section 5.3]
- o Initialize all interfaces to UNDIRECTED. [See Section 5.4]
- o Compute the DFS value, e.g. $D(x)$, and lowpoint value, $L(x)$. [See Figure 8]
- o Construct the GADAG. [See Section 5.5]
- o Assign directions to all interfaces that are still UNDIRECTED. [See Section 5.6]
- o From the computing router x , compute the next-hops for the MRT-Blue and MRT-Red. [See Section 5.7]
- o Identify alternates for each next-hop to each destination by determining which one of the blue MRT and the red MRT the computing router x should select. [See Section 5.8]

A Python implementation of this algorithm is given in Appendix A.

5.1. Interface Ordering

To ensure consistency in computation, all routers MUST order interfaces identically down to the set of links with the same metric to the same neighboring node. This is necessary for the DFS in `Lowpoint_Visit` in Section 4.3, where the selection order of the interfaces to explore results in different trees. Consistent interface ordering is also necessary for computing the GADAG, where the selection order of the interfaces to use to form ears can result in different GADAGs. It is also necessary for the topological sort described in Section 5.8, where different topological sort orderings can result in undirected links being added to the GADAG in different directions.

The required ordering between two interfaces from the same router x is given in Figure 14.

```
Interface_Compare(interface a, interface b)
  if a.metric < b.metric
    return A_LESS_THAN_B
  if b.metric < a.metric
    return B_LESS_THAN_A
  if a.neighbor.mrt_node_id < b.neighbor.mrt_node_id
    return A_LESS_THAN_B
  if b.neighbor.mrt_node_id < a.neighbor.mrt_node_id
    return B_LESS_THAN_A
  // Same metric to same node, so the order doesn't matter for
  // interoperability.
  return A_EQUAL_TO_B
```

Figure 14: Rules for ranking multiple interfaces. Order is from low to high.

In Figure 14, if two interfaces on a router connect to the same remote router with the same metric, the `Interface_Compare` function returns `A_EQUAL_TO_B`. This is because the order in which those interfaces are initially explored does not affect the final GADAG produced by the algorithm described here. While only one of the links will be added to the GADAG in the initial traversal, the other parallel links will be added to the GADAG with the same direction assigned during the procedure for assigning direction to `UNDIRECTED` links described in Section 5.6. An implementation is free to apply some additional criteria to break ties in interface ordering in this situation, but that criteria is not specified here since it will not affect the final GADAG produced by the algorithm.

The `Interface_Compare` function in Figure 14 relies on the `interface.metric` and the `interface.neighbor.mrt_node_id` values to order interfaces. The exact source of these values for different IGPs and applications is specified in Figure 15. The metric and `mrt_node_id` values for OSPFv2, OSPFv3, and IS-IS provided here is normative. The metric and `mrt_node_id` values for ISIS-PCR in this table should be considered informational. The normative values are specified in [IEEE8021Qca] .

IGP/flooding protocol and application	mrt_node_id of neighbor on interface	metric of interface
OSPFv2 for IP/LDP FRR	4 octet Neighbor Router ID in Link ID field for corresponding point-to-point link in Router-LSA	2 octet Metric field for corresponding point-to-point link in Router-LSA
OSPFv3 for IP/LDP FRR	4 octet Neighbor Router ID field for corresponding point-to-point link in Router-LSA	2 octet Metric field for corresponding point-to-point link in Router-LSA
IS-IS for IP/LDP FRR	7 octet neighbor system ID and pseudonode number in Extended IS Reachability TLV #22 or Multi-Topology IS Neighbor TLV #222	3 octet metric field in Extended IS Reachability TLV #22 or Multi-Topology IS Neighbor TLV #222
ISIS-PCR for protection of traffic in bridged networks	8 octet Bridge ID created from 2 octet Bridge Priority in SPB Instance sub-TLV (type 1) carried in MT-Capability TLV #144 and 6 octet neighbor system ID in Extended IS Reachability TLV #22 or Multi-Topology Intermediate Systems TLV #222 (informational)	3 octet SPB-LINK-METRIC in SPB-Metric sub-TLV (type 29) in Extended IS Reachability TLV #22 or Multi-Topology Intermediate Systems TLV #222. In the case of asymmetric link metrics, the larger link metric is used for both link directions. (informational)

Figure 15: value of interface.neighbor.mrt_node_id and interface.metric to be used for ranking interfaces, for different flooding protocols and applications

The metrics are unsigned integers and MUST be compared as unsigned integers. The results of `mrt_node_id` comparisons MUST be the same as would be obtained by converting the `mrt_node_ids` to unsigned integers using network byte order and performing the comparison as unsigned integers. In the case of IS-IS for IP/LDP FRR with point-to-point links, the pseudonode number (the 7th octet) is zero. Broadcast interfaces will be discussed in Section 7.

5.2. MRT Island Identification

The local MRT Island for a particular MRT profile can be determined by starting from the computing router in the network graph and doing a breadth-first-search (BFS). The BFS explores only links that are in the same area/level, are not IGP-excluded, and are not MRT-ineligible. The BFS explores only nodes that are are not IGP-excluded, and that support the particular MRT profile. See section 7 of [I-D.ietf-rtgwg-mrt-frr-architecture] for more precise definitions of these criteria.

```
MRT_Island_Identification(topology, computing_rtr, profile_id, area)
  for all routers in topology
    rtr.IN_MRT_ISLAND = FALSE
  computing_rtr.IN_MRT_ISLAND = TRUE
  explore_list = { computing_rtr }
  while (explore_list is not empty)
    next_rtr = remove_head(explore_list)
    for each intf in next_rtr
      if (not intf.MRT-ineligible
          and not intf.remote_intf.MRT-ineligible
          and not intf.IGP-excluded and (intf in area)
          and (intf.remote_node supports profile_id) )
        intf.IN_MRT_ISLAND = TRUE
        intf.remote_node.IN_MRT_ISLAND = TRUE
        if (not intf.remote_node.IN_MRT_ISLAND))
          intf.remote_node.IN_MRT_ISLAND = TRUE
          add_to_tail(explore_list, intf.remote_node)
```

Figure 16: MRT Island Identification

5.3. GADAG Root Selection

In Section 8.3 of [I-D.ietf-rtgwg-mrt-frr-architecture], the GADAG Root Selection Policy is described for the MRT default profile. This selection policy allows routers to consistently select a common GADAG Root inside the local MRT Island, based on advertised priority values. The MRT Lowpoint algorithm simply requires that all routers in the MRT Island MUST select the same GADAG Root; the mechanism can vary based upon the MRT profile description. Before beginning

computation, the network graph is reduced to contain only the set of routers that support the specific MRT profile whose MRTs are being computed.

As noted in Section 7, pseudonodes MUST NOT be considered for GADAG root selection.

It is expected that an operator will designate a set of routers as good choices for selection as GADAG root by setting the GADAG Root Selection Priority for that set of routers to lower (more preferred) numerical values. For guidance on setting the GADAG Root Selection Priority values, refer to Section 10.1.

5.4. Initialization

Before running the algorithm, there is the standard type of initialization to be done, such as clearing any computed DFS-values, lowpoint-values, DFS-parents, lowpoint-parents, any MRT-computed next-hops, and flags associated with algorithm.

It is assumed that a regular SPF computation has been run so that the primary next-hops from the computing router to each destination are known. This is required for determining alternates at the last step.

Initially, all interfaces MUST be initialized to UNDIRECTED. Whether they are OUTGOING, INCOMING or both is determined when the GADAG is constructed and augmented.

It is possible that some links and nodes will be marked using standard IGP mechanisms to discourage or prevent transit traffic. Section 7.3.1 of [I-D.ietf-rtgwg-mrt-frr-architecture] describes how those links and nodes are excluded from MRT Island formation.

MRT-FRR also has the ability to advertise links MRT-Ineligible, as described in Section 7.3.2 of [I-D.ietf-rtgwg-mrt-frr-architecture]. These links are excluded from the MRT Island and the GADAG. Computation of MRT next-hops will therefore not use any MRT-ineligible links. The MRT algorithm does still need to consider MRT-ineligible links when computing FRR alternates, because an MRT-ineligible link can still be the shortest-path next-hop to reach a destination.

When a broadcast interface is advertised as MRT-ineligible, then the pseudo-node representing the entire broadcast network MUST NOT be included in the MRT Island. This is equivalent to excluding all of the broadcast interfaces on that broadcast network from the MRT Island.

5.5. Constructing the GADAG using lowpoint inheritance

As discussed in Section 4.2, it is necessary to find ears from a node x that is already in the GADAG (known as IN_GADAG). Two different methods are used to find ears in the algorithm. The first is by going to a not IN_GADAG DFS-child and then following the chain of low-point parents until an IN_GADAG node is found. The second is by going to a not IN_GADAG neighbor and then following the chain of DFS parents until an IN_GADAG node is found. As an ear is found, the associated interfaces are marked based on the direction taken. The nodes in the ear are marked as IN_GADAG. In the algorithm, first the ears via DFS-children are found and then the ears via DFS-neighbors are found.

By adding both types of ears when an IN_GADAG node is processed, all ears that connect to that node are found. The order in which the IN_GADAG nodes is processed is, of course, key to the algorithm. The order is a stack of ears so the most recent ear is found at the top of the stack. Of course, the stack stores nodes and not ears, so an ordered list of nodes, from the first node in the ear to the last node in the ear, is created as the ear is explored and then that list is pushed onto the stack.

Each ear represents a partial order (see Figure 4) and processing the nodes in order along each ear ensures that all ears connecting to a node are found before a node higher in the partial order has its ears explored. This means that the direction of the links in the ear is always from the node x being processed towards the other end of the ear. Additionally, by using a stack of ears, this means that any unprocessed nodes in previous ears can only be ordered higher than nodes in the ears below it on the stack.

In this algorithm that depends upon Low-Point inheritance, it is necessary that every node have a low-point parent that is not itself. If a node is a cut-vertex, that may not yet be the case. Therefore, any nodes without a low-point parent will have their low-point parent set to their DFS parent and their low-point value set to the DFS-value of their parent. This assignment also properly allows an ear between two cut-vertices.

Finally, the algorithm simultaneously computes each node's local-root, as described in Figure 12. This is further elaborated as follows. The local-root can be inherited from the node at the end of the ear unless the end of the ear is x itself, in which case the local-root for all the nodes in the ear would be x . This is because whenever the first cycle is found in a block, or an ear involving a bridge is computed, the cut-vertex closest to the root would be x itself. In all other scenarios, the properties of lowpoint/dfs

parents ensure that the end of the ear will be in the same block, and thus inheriting its local-root would be the correct local-root for all newly added nodes.

The pseudo-code for the GADAG algorithm (assuming that the adjustment of lowpoint for cut-vertices has been made) is shown in Figure 17.

```

Construct_Ear(x, Stack, intf, ear_type)
    ear_list = empty
    cur_node = intf.remote_node
    cur_intf = intf
    not_done = true

    while not_done
        cur_intf.UNDIRECTED = false
        cur_intf.OUTGOING = true
        cur_intf.remote_intf.UNDIRECTED = false
        cur_intf.remote_intf.INCOMING = true

        if cur_node.IN_GADAG is false
            cur_node.IN_GADAG = true
            add_to_list_end(ear_list, cur_node)
            if ear_type is CHILD
                cur_intf = cur_node.lowpoint_parent_intf
                cur_node = cur_node.lowpoint_parent
            else // ear_type must be NEIGHBOR
                cur_intf = cur_node.dfs_parent_intf
                cur_node = cur_node.dfs_parent
        else
            not_done = false

    if (ear_type is CHILD) and (cur_node is x)
        // x is a cut-vertex and the local root for
        // the block in which the ear is computed
        x.IS_CUT_VERTEX = true
        localroot = x
    else
        // Inherit local-root from the end of the ear
        localroot = cur_node.localroot
    while ear_list is not empty
        y = remove_end_item_from_list(ear_list)
        y.localroot = localroot
        push(Stack, y)

Construct_GADAG_via_Lowpoint(topology, gadag_root)
    gadag_root.IN_GADAG = true
    gadag_root.localroot = None
    Initialize Stack to empty

```

```
push gadag_root onto Stack
while (Stack is not empty)
  x = pop(Stack)
  foreach ordered_interface intf of x
    if ((intf.remote_node.IN_GADAG == false) and
        (intf.remote_node.dfs_parent is x))
      Construct_Ear(x, Stack, intf, CHILD)
  foreach ordered_interface intf of x
    if ((intf.remote_node.IN_GADAG == false) and
        (intf.remote_node.dfs_parent is not x))
      Construct_Ear(x, Stack, intf, NEIGHBOR)

Construct_GADAG_via_Lowpoint(topology, gadag_root)
```

Figure 17: Low-point Inheritance GADAG algorithm

5.6. Augmenting the GADAG by directing all links

The GADAG, regardless of the method used to construct it, at this point could be used to find MRTs, but the topology does not include all links in the network graph. That has two impacts. First, there might be shorter paths that respect the GADAG partial ordering and so the alternate paths would not be as short as possible. Second, there may be additional paths between a router *x* and the root that are not included in the GADAG. Including those provides potentially more bandwidth to traffic flowing on the alternates and may reduce congestion compared to just using the GADAG as currently constructed.

The goal is thus to assign direction to every remaining link marked as UNDIRECTED to improve the paths and number of paths found when the MRTs are computed.

To do this, we need to establish a total order that respects the partial order described by the GADAG. This can be done using Kahn's topological sort [Kahn_1962_topo_sort] which essentially assigns a number to a node *x* only after all nodes before it (e.g. with a link incoming to *x*) have had their numbers assigned. The only issue with the topological sort is that it works on DAGs and not ADAGs or GADAGs.

To convert a GADAG to a DAG, it is necessary to remove all links that point to a root of block from within that block. That provides the necessary conversion to a DAG and then a topological sort can be done. When adding undirected links to the GADAG, links connecting the block root to other nodes in that block need special handling because the topological order will not always give the right answer for those links. There are three cases to consider. If the undirected link in question has another parallel link between the

same two nodes that is already directed, then the direction of the undirected link can be inherited from the previously directed link. In the case of parallel cut links, we set all of the parallel links to both INCOMING and OUTGOING. Otherwise, the undirected link in question is set to OUTGOING from the block root node. A cut-link can then be identified by the fact that it will be directed both INCOMING and OUTGOING in the GADAG. The exact details of this whole process are captured in Figure 18

```

Add_Undirected_Block_Root_Links(topo, gadag_root)
  foreach node x in topo
    if x.IS_CUT_VERTEX or x is gadag_root
      foreach interface i of x
        if (i.remote_node.localroot is not x
            or i.PROCESSED )
          continue
        Initialize bundle_list to empty
        bundle.UNDIRECTED = true
        bundle.OUTGOING = false
        bundle.INCOMING = false
        foreach interface i2 in x
          if i2.remote_node is i.remote_node
            add_to_list_end(bundle_list, i2)
            if not i2.UNDIRECTED:
              bundle.UNDIRECTED = false
              if i2.INCOMING:
                bundle.INCOMING = true
              if i2.OUTGOING:
                bundle.OUTGOING = true
        if bundle.UNDIRECTED
          foreach interface i3 in bundle_list
            i3.UNDIRECTED = false
            i3.remote_intf.UNDIRECTED = false
            i3.PROCESSED = true
            i3.remote_intf.PROCESSED = true
            i3.OUTGOING = true
            i3.remote_intf.INCOMING = true
        else
          if (bundle.OUTGOING and bundle.INCOMING)
            foreach interface i3 in bundle_list
              i3.UNDIRECTED = false
              i3.remote_intf.UNDIRECTED = false
              i3.PROCESSED = true
              i3.remote_intf.PROCESSED = true
              i3.OUTGOING = true
              i3.INCOMING = true
              i3.remote_intf.INCOMING = true
              i3.remote_intf.OUTGOING = true

```

```

else if bundle.OUTGOING
    foreach interface i3 in bundle_list
        i3.UNDIRECTED = false
        i3.remote_intf.UNDIRECTED = false
        i3.PROCESSED = true
        i3.remote_intf.PROCESSED = true
        i3.OUTGOING = true
        i3.remote_intf.INCOMING = true
else if bundle.INCOMING
    foreach interface i3 in bundle_list
        i3.UNDIRECTED = false
        i3.remote_intf.UNDIRECTED = false
        i3.PROCESSED = true
        i3.remote_intf.PROCESSED = true
        i3.INCOMING = true
        i3.remote_intf.OUTGOING = true

Modify_Block_Root_Incoming_Links(topo, gadag_root)
foreach node x in topo
    if x.IS_CUT_VERTEX or x is gadag_root
        foreach interface i of x
            if i.remote_node.localroot is x
                if i.INCOMING:
                    i.INCOMING = false
                    i.INCOMING_STORED = true
                    i.remote_intf.OUTGOING = false
                    i.remote_intf.OUTGOING_STORED = true

Revert_Block_Root_Incoming_Links(topo, gadag_root)
foreach node x in topo
    if x.IS_CUT_VERTEX or x is gadag_root
        foreach interface i of x
            if i.remote_node.localroot is x
                if i.INCOMING_STORED
                    i.INCOMING = true
                    i.remote_intf.OUTGOING = true
                    i.INCOMING_STORED = false
                    i.remote_intf.OUTGOING_STORED = false

Run_Topological_Sort_GADAG(topo, gadag_root)
Modify_Block_Root_Incoming_Links(topo, gadag_root)
foreach node x in topo
    node.unvisited = 0
    foreach interface i of x
        if (i.INCOMING)
            node.unvisited += 1
Initialize working_list to empty
Initialize topo_order_list to empty

```

```

add_to_list_end(working_list, gadag_root)
while working_list is not empty
  y = remove_start_item_from_list(working_list)
  add_to_list_end(topo_order_list, y)
  foreach ordered_interface i of y
    if intf.OUTGOING
      i.remote_node.unvisited -= 1
      if i.remote_node.unvisited is 0
        add_to_list_end(working_list, i.remote_node)
next_topo_order = 1
while topo_order_list is not empty
  y = remove_start_item_from_list(topo_order_list)
  y.topo_order = next_topo_order
  next_topo_order += 1
Revert_Block_Root_Incoming_Links(topo, gadag_root)

def Set_Other_Undirected_Links_Based_On_Topo_Order(topo)
  foreach node x in topo
    foreach interface i of x
      if i.UNDIRECTED:
        if x.topo_order < i.remote_node.topo_order
          i.OUTGOING = true
          i.UNDIRECTED = false
          i.remote_intf.INCOMING = true
          i.remote_intf.UNDIRECTED = false
        else
          i.INCOMING = true
          i.UNDIRECTED = false
          i.remote_intf.OUTGOING = true
          i.remote_intf.UNDIRECTED = false

Add_Undirected_Links(topo, gadag_root)
Add_Undirected_Block_Root_Links(topo, gadag_root)
Run_Topological_Sort_GADAG(topo, gadag_root)
Set_Other_Undirected_Links_Based_On_Topo_Order(topo)

Add_Undirected_Links(topo, gadag_root)

```

Figure 18: Assigning direction to UNDIRECTED links

Proxy-nodes do not need to be added to the network graph. They cannot be transited and do not affect the MRTs that are computed. The details of how the MRT-Blue and MRT-Red next-hops are computed for proxy-nodes and how the appropriate alternate next-hops are selected is given in Section 5.9.

5.7. Compute MRT next-hops

As was discussed in Section 4.1, once a ADAG is found, it is straightforward to find the next-hops from any node X to the ADAG root. However, in this algorithm, we will reuse the common GADAG and find not only the one pair of MRTs rooted at the GADAG root with it, but find a pair rooted at each node. This is useful since it is significantly faster to compute.

The method for computing differently rooted MRTs from the common GADAG is based on two ideas. First, if two nodes X and Y are ordered with respect to each other in the partial order, then an SPF along OUTGOING links (an increasing-SPF) and an SPF along INCOMING links (a decreasing-SPF) can be used to find the increasing and decreasing paths. Second, if two nodes X and Y aren't ordered with respect to each other in the partial order, then intermediary nodes can be used to create the paths by increasing/decreasing to the intermediary and then decreasing/increasing to reach Y.

As usual, the two basic ideas will be discussed assuming the network is two-connected. The generalization to multiple blocks is discussed in Section 5.7.4. The full algorithm is given in Section 5.7.5.

5.7.1. MRT next-hops to all nodes ordered with respect to the computing node

To find two node-disjoint paths from the computing router X to any node Y, depends upon whether $Y \gg X$ or $Y \ll X$. As shown in Figure 19, if $Y \gg X$, then there is an increasing path that goes from X to Y without crossing R; this contains nodes in the interval $[X, Y]$. There is also a decreasing path that decreases towards R and then decreases from R to Y; this contains nodes in the interval $[X, R\text{-small}]$ or $[R\text{-great}, Y]$. The two paths cannot have common nodes other than X and Y.

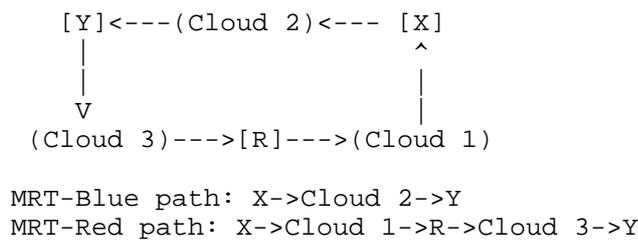


Figure 19: $Y \gg X$

Similar logic applies if $Y \ll X$, as shown in Figure 20. In this case, the increasing path from X increases to R and then increases from R to Y to use nodes in the intervals $[X, R\text{-great}]$ and $[R\text{-small}, Y]$. The decreasing path from X reaches Y without crossing R and uses nodes in the interval $[Y, X]$.

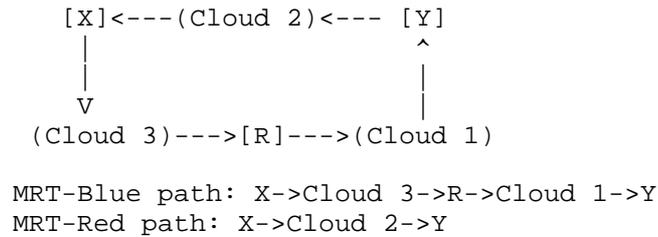


Figure 20: $Y \ll X$

5.7.2. MRT next-hops to all nodes not ordered with respect to the computing node

When X and Y are not ordered, the first path should increase until we get to a node G, where $G \gg Y$. At G, we need to decrease to Y. The other path should be just the opposite: we must decrease until we get to a node H, where $H \ll Y$, and then increase. Since R is smaller and greater than Y, such G and H must exist. It is also easy to see that these two paths must be node disjoint: the first path contains nodes in interval $[X, G]$ and $[Y, G]$, while the second path contains nodes in interval $[H, X]$ and $[H, Y]$. This is illustrated in Figure 21. It is necessary to decrease and then increase for the MRT-Blue and increase and then decrease for the MRT-Red; if one simply increased for one and decreased for the other, then both paths would go through the root R.

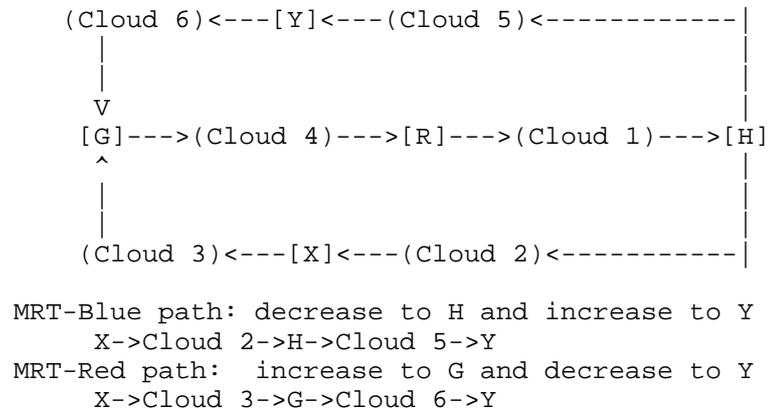


Figure 21: X and Y unordered

This gives disjoint paths as long as G and H are not the same node. Since $G \gg Y$ and $H \ll Y$, if G and H could be the same node, that would have to be the root R. This is not possible because there is only one incoming interface to the root R which is created when the initial cycle is found. Recall from Figure 6 that whenever an ear was found to have an end that was the root R, the ear was directed from R so that the associated interface on R is outgoing and not incoming. Therefore, there must be exactly one node M which is the largest one before R, so the MRT-Red path will never reach R; it will turn at M and decrease to Y.

5.7.3. Computing Redundant Tree next-hops in a 2-connected Graph

The basic ideas for computing RT next-hops in a 2-connected graph were given in Section 5.7.1 and Section 5.7.2. Given these two ideas, how can we find the trees?

If some node X only wants to find the next-hops (which is usually the case for IP networks), it is enough to find which nodes are greater and less than X, and which are not ordered; this can be done by running an increasing-SPF and a decreasing-SPF rooted at X and not exploring any links from the ADAG root.

In principle, a traversal method other than SPF could be used to traverse the GADAG in the process of determining blue and red next-hops that result in maximally redundant trees. This will be the case as long as one traversal uses the links in the direction specified by the GADAG and the other traversal uses the links in the direction opposite of that specified by the GADAG. However, a different traversal algorithm will generally result in different blue and red next-hops. Therefore, the algorithm specified here requires the use

of SPF to traverse the GADAG to generate MRT blue and red next-hops, as described below.

An increasing-SPF rooted at X and not exploring links from the root will find the increasing next-hops to all $Y \gg X$. Those increasing next-hops are X's next-hops on the MRT-Blue to reach Y. A decreasing-SPF rooted at X and not exploring links from the root will find the decreasing next-hops to all $Z \ll X$. Those decreasing next-hops are X's next-hops on the MRT-Red to reach Z. Since the root R is both greater than and less than X, after this increasing-SPF and decreasing-SPF, X's next-hops on the MRT-Blue and on the MRT-Red to reach R are known. For every node $Y \gg X$, X's next-hops on the MRT-Red to reach Y are set to those on the MRT-Red to reach R. For every node $Z \ll X$, X's next-hops on the MRT-Blue to reach Z are set to those on the MRT-Blue to reach R.

For those nodes which were not reached by either the increasing-SPF or the decreasing-SPF, we can determine the next-hops as well. The increasing MRT-Blue next-hop for a node which is not ordered with respect to X is the next-hop along the decreasing MRT-Red towards R, and the decreasing MRT-Red next-hop is the next-hop along the increasing MRT-Blue towards R. Naturally, since R is ordered with respect to all the nodes, there will always be an increasing and a decreasing path towards it. This algorithm does not provide the complete specific path taken but just the appropriate next-hops to use. The identities of G and H are not determined by the computing node X.

The final case to consider is when the GADAG root R computes its own next-hops. Since the GADAG root R is \ll all other nodes, running an increasing-SPF rooted at R will reach all other nodes; the MRT-Blue next-hops are those found with this increasing-SPF. Similarly, since the GADAG root R is \gg all other nodes, running a decreasing-SPF rooted at R will reach all other nodes; the MRT-Red next-hops are those found with this decreasing-SPF.

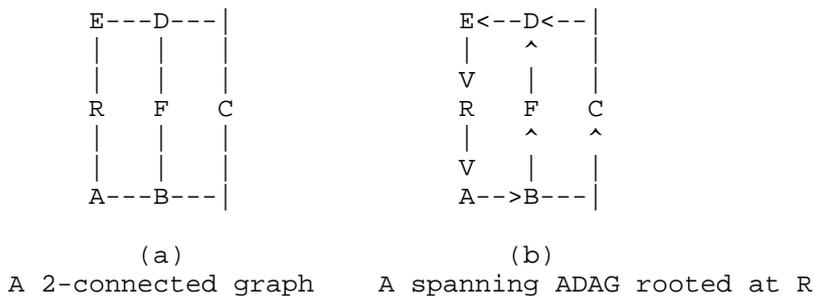


Figure 22

As an example consider the situation depicted in Figure 22. Node C runs an increasing-SPF and a decreasing-SPF on the ADAG. The increasing-SPF reaches D, E and R and the decreasing-SPF reaches B, A and R. $E \gg C$. So towards E the MRT-Blue next-hop is D, since E was reached on the increasing path through D. And the MRT-Red next-hop towards E is B, since R was reached on the decreasing path through B. Since $E \gg D$, D will similarly compute its MRT-Blue next-hop to be E, ensuring that a packet on MRT-Blue will use path C-D-E. B, A and R will similarly compute the MRT-Red next-hops towards E (which is ordered less than B, A and R), ensuring that a packet on MRT-Red will use path C-B-A-R-E.

C can determine the next-hops towards F as well. Since F is not ordered with respect to C, the MRT-Blue next-hop is the decreasing one towards R (which is B) and the MRT-Red next-hop is the increasing one towards R (which is D). Since $F \gg B$, for its MRT-Blue next-hop towards F, B will use the real increasing next-hop towards F. So a packet forwarded to B on MRT-Blue will get to F on path C-B-F. Similarly, D will use the real decreasing next-hop towards F as its MRT-Red next-hop, a packet on MRT-Red will use path C-D-F.

5.7.4. Generalizing for a graph that isn't 2-connected

If a graph isn't 2-connected, then the basic approach given in Section 5.7.3 needs some extensions to determine the appropriate MRT next-hops to use for destinations outside the computing router X's blocks. In order to find a pair of maximally redundant trees in that graph we need to find a pair of RTs in each of the blocks (the root of these trees will be discussed later), and combine them.

When computing the MRT next-hops from a router X, there are three basic differences:

1. Only nodes in a common block with X should be explored in the increasing-SPF and decreasing-SPF.
2. Instead of using the GADAG root, X's local-root should be used. This has the following implications:
 - A. The links from X's local-root should not be explored.
 - B. If a node is explored in the outgoing SPF so $Y \gg X$, then X's MRT-Red next-hops to reach Y uses X's MRT-Red next-hops to reach X's local-root and if $Z \ll X$, then X's MRT-Blue next-hops to reach Z uses X's MRT-Blue next-hops to reach X's local-root.

- C. If a node W in a common block with X was not reached in the increasing-SPF or decreasing-SPF, then W is unordered with respect to X. X's MRT-Blue next-hops to W are X's decreasing (aka MRT-Red) next-hops to X's local-root. X's MRT-Red next-hops to W are X's increasing (aka MRT-Blue) next-hops to X's local-root.
3. For nodes in different blocks, the next-hops must be inherited via the relevant cut-vertex.

These are all captured in the detailed algorithm given in Section 5.7.5.

5.7.5. Complete Algorithm to Compute MRT Next-Hops

The complete algorithm to compute MRT Next-Hops for a particular router X is given in Figure 23. In addition to computing the MRT-Blue next-hops and MRT-Red next-hops used by X to reach each node Y, the algorithm also stores an "order_proxy", which is the proper cut-vertex to reach Y if it is outside the block, and which is used later in deciding whether the MRT-Blue or the MRT-Red can provide an acceptable alternate for a particular primary next-hop.

```

In_Common_Block(x, y)
  if ( (x.block_id is y.block_id)
      or (x is y.localroot) or (y is x.localroot) )
    return true
  return false

Store_Results(y, direction)
  if direction is FORWARD
    y.higher = true
    y.blue_next_hops = y.next_hops
  if direction is REVERSE
    y.lower = true
    y.red_next_hops = y.next_hops

SPF_No_Traverse_Block_Root(spfx_root, block_root, direction)
  Initialize spfx_heap to empty
  Initialize nodes' spfx_metric to infinity and next_hops to empty
  spfx_root.spfx_metric = 0
  insert(spfx_heap, spfx_root)
  while (spfx_heap is not empty)
    min_node = remove_lowest(spfx_heap)
    Store_Results(min_node, direction)
    if ((min_node is spfx_root) or (min_node is not block_root))
      foreach interface intf of min_node
        if ( ( (direction is FORWARD) and intf.OUTGOING) or

```

```

        ((direction is REVERSE) and intf.INCOMING) )
        and In_Common_Block(spf_root, intf.remote_node) )
path_metric = min_node.spf_metric + intf.metric
if path_metric < intf.remote_node.spf_metric
    intf.remote_node.spf_metric = path_metric
    if min_node is spf_root
        intf.remote_node.next_hops = make_list(intf)
    else
        intf.remote_node.next_hops = min_node.next_hops
        insert_or_update(spf_heap, intf.remote_node)
else if path_metric == intf.remote_node.spf_metric
    if min_node is spf_root
        add_to_list(intf.remote_node.next_hops, intf)
    else
        add_list_to_list(intf.remote_node.next_hops,
                        min_node.next_hops)

SetEdge(y)
    if y.blue_next_hops is empty and y.red_next_hops is empty
        SetEdge(y.localroot)
        y.blue_next_hops = y.localroot.blue_next_hops
        y.red_next_hops = y.localroot.red_next_hops
        y.order_proxy = y.localroot.order_proxy

Compute_MRT_NextHops(x, gadag_root)
    foreach node y
        y.higher = y.lower = false
        clear y.red_next_hops and y.blue_next_hops
        y.order_proxy = y
    SPF_No_Traverse_Block_Root(x, x.localroot, FORWARD)
    SPF_No_Traverse_Block_Root(x, x.localroot, REVERSE)

    // red and blue next-hops are stored to x.localroot as different
    // paths are found via the SPF and reverse-SPF.
    // Similarly any nodes whose local-root is x will have their
    // red_next_hops and blue_next_hops already set.

    // Handle nodes in the same block that aren't the local-root
    foreach node y
        if (y.IN_MRT_ISLAND and (y is not x) and
            (y.block_id is x.block_id) )
            if y.higher
                y.red_next_hops = x.localroot.red_next_hops
            else if y.lower
                y.blue_next_hops = x.localroot.blue_next_hops
            else
                y.blue_next_hops = x.localroot.red_next_hops
                y.red_next_hops = x.localroot.blue_next_hops

```

```

// Inherit next-hops and order_proxies to other components
if (x is not gadag_root) and (x.localroot is not gadag_root)
    gadag_root.blue_next_hops = x.localroot.blue_next_hops
    gadag_root.red_next_hops = x.localroot.red_next_hops
    gadag_root.order_proxy = x.localroot
foreach node y
    if (y is not gadag_root) and (y is not x) and y.IN_MRT_ISLAND
        SetEdge(y)

max_block_id = 0
Assign_Block_ID(gadag_root, max_block_id)
Compute_MRT_NextHops(x, gadag_root)

```

Figure 23

5.8. Identify MRT alternates

At this point, a computing router *S* knows its MRT-Blue next-hops and MRT-Red next-hops for each destination in the MRT Island. The primary next-hops along the SPT are also known. It remains to determine for each primary next-hop to a destination *D*, which of the MRTs avoids the primary next-hop node *F*. This computation depends upon data set in `Compute_MRT_NextHops` such as each node *y*'s `y.blue_next_hops`, `y.red_next_hops`, `y.order_proxy`, `y.higher`, `y.lower` and `topo_orders`. Recall that any router knows only which are the nodes greater and lesser than itself, but it cannot decide the relation between any two given nodes easily; that is why we need topological ordering.

For each primary next-hop node *F* to each destination *D*, *S* can call `Select_Alternates(S, D, F, primary_intf)` to determine whether to use the MRT-Blue or MRT-Red next-hops as the alternate next-hop(s) for that primary next hop. The algorithm is given in Figure 24 and discussed afterwards.

```

Select_Alternates_Internal(D, F, primary_intf,
                          D_lower, D_higher, D_topo_order):
    if D_higher and D_lower
        if F.HIGHER and F.LOWER
            if F.topo_order < D_topo_order
                return USE_RED
            else
                return USE_BLUE
        if F.HIGHER
            return USE_RED
        if F.LOWER
            return USE_BLUE
    //F unordered wrt S

```

```
        return USE_RED_OR_BLUE

    else if D_higher
        if F.HIGHER and F.LOWER
            return USE_BLUE
        if F.LOWER
            return USE_BLUE
        if F.HIGHER
            if (F.topo_order > D_topo_order)
                return USE_BLUE
            if (F.topo_order < D_topo_order)
                return USE_RED
        //F unordered wrt S
        return USE_RED_OR_BLUE

    else if D_lower
        if F.HIGHER and F.LOWER
            return USE_RED
        if F.HIGHER
            return USE_RED
        if F.LOWER
            if F.topo_order > D_topo_order
                return USE_BLUE
            if F.topo_order < D_topo_order
                return USE_RED
        //F unordered wrt S
        return USE_RED_OR_BLUE

    else //D is unordered wrt S
        if F.HIGHER and F.LOWER
            if primary_intf.OUTGOING and primary_intf.INCOMING
                return USE_RED_OR_BLUE
            if primary_intf.OUTGOING
                return USE_BLUE
            if primary_intf.INCOMING
                return USE_RED
            //primary_intf not in GADAG
            return USE_RED
        if F.LOWER
            return USE_RED
        if F.HIGHER
            return USE_BLUE
        //F unordered wrt S
        if F.topo_order > D_topo_order:
            return USE_BLUE
        else:
            return USE_RED
```

```

Select_Alternates(D, F, primary_intf)
  if not In_Common_Block(F, S)
    return PRIM_NH_IN_DIFFERENT_BLOCK
  if (D is F) or (D.order_proxy is F)
    return PRIM_NH_IS_D_OR_OP_FOR_D
  D_lower = D.order_proxy.LOWER
  D_higher = D.order_proxy.HIGHER
  D_topo_order = D.order_proxy.topo_order
  return Select_Alternates_Internal(D, F, primary_intf,
    D_lower, D_higher, D_topo_order)

```

Figure 24: Select_Alternates() and Select_Alternates_Internal()

It is useful to first handle the case where where F is also D, or F is the order proxy for D. In this case, only link protection is possible. The MRT that doesn't use the failed primary next-hop is used. If both MRTs use the primary next-hop, then the primary next-hop must be a cut-link, so either MRT could be used but the set of MRT next-hops must be pruned to avoid the failed primary next-hop interface. To indicate this case, Select_Alternates returns PRIM_NH_IS_D_OR_OP_FOR_D. Explicit pseudo-code to handle the three sub-cases above is not provided.

The logic behind Select_Alternates_Internal is described in Figure 25. As an example, consider the first case described in the table, where the $D \gg S$ and $D \ll S$. If this is true, then either S or D must be the block root, R. If $F \gg S$ and $F \ll S$, then S is the block root. So the blue path from S to D is the increasing path to D, and the red path S to D is the decreasing path to D. If the $F.topo_order < D.topo_order$, then either F is ordered higher than D or F is unordered with respect to D. Therefore, F is either on a decreasing path from S to D, or it is on neither an increasing nor a decreasing path from S to D. In either case, it is safe to take an increasing path from S to D to avoid F. We know that when S is R, the increasing path is the blue path, so it is safe to use the blue path to avoid F.

If instead $F.topo_order > D.topo_order$, then either F is ordered lower than D, or F is unordered with respect to D. Therefore, F is either on an increasing path from S to D, or it is on neither an increasing nor a decreasing path from S to D. In either case, it is safe to take a decreasing path from S to D to avoid F. We know that when S is R, the decreasing path is the red path, so it is safe to use the red path to avoid F.

If $F \gg S$ or $F \ll S$ (but not both), then D is the block root. We then know that the blue path from S to D is the increasing path to R, and the red path is the decreasing path to R. When $F \gg S$, we deduce that

F is on an increasing path from S to R. So in order to avoid F, we use a decreasing path from S to R, which is the red path. Instead, when $F \ll S$, we deduce that F is on a decreasing path from S to R. So in order to avoid F, we use an increasing path from S to R, which is the blue path.

All possible cases are systematically described in the same manner in the rest of the table.

D wrt S	MRT blue and red path properties	F wrt S	additional criteria	F wrt MRT (deduced)	Alternate
D >> S and D << S, D is R, or S is R	Blue path: Increasing path to R. Red path: Decreasing path to R.	F >> S only	additional criteria not needed	F on an increasing path from S to R	Use Red to avoid F
		F << S only	additional criteria not needed	F on a decreasing path from S to R	Use Blue to avoid F
		F >> S and F << S, F is R	topo(F) > topo(D) implies that F >> D or F ?? D	F on a decreasing path from S to D or neither	Use Blue to avoid F
			topo(F) < topo(D) implies that F << D or F ?? D	F on an increasing path from S to D or neither	Use Red to avoid F
		F ?? S	Can only occur when link between F and S is marked MRT_INELIGIBLE	F is on neither increasing nor decr. path from S to D or R	Use Red or Blue to avoid F
D >> S only	Blue path: Increasing shortest path from	F << S only	additional criteria not needed	F on decreasing path from S to R	Use Blue to avoid F

	S to D. Red path: Decreasing shortest path from S to R, then decreasing shortest path from R to D.	F>>S only	topo(F)>topo(D) implies that F>>D or F??D	F on decreasing path from R to D or neither	Use Blue to avoid F
			topo(F)<topo(D) implies that F<<D or F??D	F on increasing path from S to D or neither	Use Red to avoid F
		F>>S and F<<S, F is R	additional criteria not needed	F on Red	Use Blue to avoid F
		F??S	Can only occur when link between F and S is marked MRT_INELIGIBLE	F is on neither increasing nor decr. path from S to D or R	Use Red or Blue to avoid F
D<<S only	Blue path: Increasing shortest path from S to R, then increasing shortest path from R to D. Red path: Decreasing shortest path from S to D.	F>>S only	additional criteria not needed	F on increasing path from S to R	Use Red to avoid F
		F<<S only	topo(F)>topo(D) implies that F>>D or F??D	F on decreasing path from R to D or neither	Use Blue to avoid F
			topo(F)<topo(D) implies that F<<D or F??D	F on increasing path from S to D or neither	Use Red to avoid F
		F>>S	additional	F on Blue	Use Red

		and F<<S, F is R	criteria not needed		to avoid F	
		F??S	Can only occur when link between F and S is marked MRT_INELIGIBLE	F is on neither increasing nor decr. path from S to D or R	Use Red or Blue to avoid F	
D??S	Blue path: Decr. from S to first node K<<D, then incr. to D. Red path: Incr. from S to first node L>>D, then decr.	F<<S only	additional criteria not needed	F on a decreasing path from S to K.	Use Red to avoid F	
		F>>S only	additional criteria not needed	F on an increasing path from S to L	Use Blue to avoid F	
		F??S	F<-->S link is MRT_INELIGIBLE			
				topo(F)>topo(D) implies that F>>D or F??D	F on decr. path from L to D or neither	Use Blue to avoid F
				topo(F)<topo(D) implies that F<<D or F??D	F on incr. path from K to D or neither	Use Red to avoid F
		F>>S and F<<S, F is R	GADAG link direction S->F		F on an incr. path from S	Use Blue to avoid F
			GADAG link direction S<-F		F on a decr. path from S	Use Red to avoid F
GADAG link direction S<-->F			Either F is the order proxy for D (case already handled) or D			

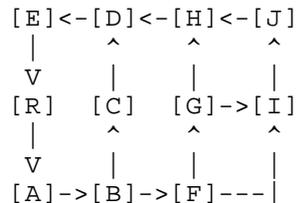
				is in a different block from F, in which case Red or Blue avoids F
			S-F link not in GADAG, only when S-F link is MRT_INELIGIBLE	Relies on special construction of GADAG to demonstrate that using Red avoids F (see text)

Figure 25: determining MRT next-hops and alternates based on the partial order and topological sort relationships between the source(S), destination(D), primary next-hop(F), and block root(R). topo(N) indicates the topological sort value of node N. X??Y indicates that node X is unordered with respect to node Y. It is assumed that the case where F is D, or where F is the order proxy for D, has already been handled.

The last case in Figure 25 requires additional explanation. The fact that the red path from S to D in this case avoids F relies on a special property of the GADAGs that we have constructed in this algorithm, a property not shared by all GADAGs in general. When D is unordered with respect to S, and F is the localroot for S, it can occur that the link between S and F is not in the GADAG only when that link has been marked MRT_INELIGIBLE. For an arbitrary GADAG, S doesn't have enough information based on the computed order relationships to determine if the red path or blue path will hit F (which is also the localroot) before hitting K or L, and making it safely to D. However, the GADAGs that we construct using the algorithm in this document are not arbitrary GADAGs. They have the additional property that incoming links to a localroot come from only one other node in the same block. This is a result of the method of construction. This additional property guarantees that the red path from S to D will never pass through the localroot of S. (That would require the localroot to play the role of L, the first node in the path ordered higher than D, which would in turn require the localroot to have two incoming links in the GADAG, which cannot happen.) Therefore it is safe to use the red path to avoid F with these specially constructed GADAGs.

As an example of how `Select_Alternates_Internal()` operates, consider the ADAG depicted in Figure 26 and first suppose that G is the source, D is the destination and H is the failed next-hop. Since $D \gg G$, we need to compare `H.topo_order` and `D.topo_order`. Since $D.topo_order > H.topo_order$, D must be either higher than H or unordered with respect to H, so we should select the decreasing path towards the root. If, however, the destination were instead J, we

must find that $H.topo_order > J.topo_order$, so we must choose the increasing Blue next-hop to J , which is I . In the case, when instead the destination is C , we find that we need to first decrease to avoid using H , so the Blue, first decreasing then increasing, path is selected.



(a)ADAG rooted at R for
a 2-connected graph

Figure 26

5.9. Named Proxy-Nodes

As discussed in Section 11.2 of [I-D.ietf-rtgwg-mrt-frr-architecture], it is necessary to find MRT-Blue and MRT-Red next-hops and MRT-FRR alternates for named proxy-nodes. An example use case is for a router that is not part of that local MRT Island, when there is only partial MRT support in the domain.

5.9.1. Determining Proxy-Node Attachment Routers

Section 11.2 of [I-D.ietf-rtgwg-mrt-frr-architecture] discusses general considerations for determining the two proxy-node attachment routers for a given proxy-node, corresponding to a prefix. A router in the MRT Island that advertises the prefix is a candidate for being a proxy-node attachment router, with the associated named-proxy-cost equal to the advertised cost to the prefix.

An Island Border Router (IBR) is a router in the MRT Island that is connected to an Island Neighbor(IN), which is a router not in the MRT Island but in the same area/level. An (IBR,IN) pair is a candidate for being a proxy-node attachment router, if the shortest path from the IN to the prefix does not enter the MRT Island. A method for identifying such loop-free Island Neighbors(LFINs) is given below. The named-proxy-cost assigned to each (IBR, IN) pair is $cost(IBR, IN) + D_{opt}(IN, prefix)$.

From the set of prefix-advertising routers and the set of IBRs with at least one LFIN, the two routers with the lowest named-proxy-cost

are selected. Ties are broken based upon the lowest Router ID. For ease of discussion, the two selected routers will be referred to as proxy-node attachment routers.

5.9.2. Computing if an Island Neighbor (IN) is loop-free

As discussed above, the Island Neighbor needs to be loop-free with respect to the whole MRT Island for the destination. This can be accomplished by running the usual SPF algorithm while keeping track of which shortest paths have passed through the MRT island. Pseudo-code for this is shown in Figure 27. The `Island_Marking_SPF()` is run for each IN that needs to be evaluated for the loop-free condition, with the IN as the `spf_root`. Whether or not an IN is loop-free with respect to the MRT island can then be determined by evaluating `node.PATH_HITS_ISLAND` for each destination of interest.

```

Island_Marking_SPF(sp_f_root)
  Initialize spf_heap to empty
  Initialize nodes' spf_metric to infinity and next_hops to empty
  and PATH_HITS_ISLAND to false
  spf_root.spf_metric = 0
  insert(spf_heap, spf_root)
  while (spf_heap is not empty)
    min_node = remove_lowest(spf_heap)
    foreach interface intf of min_node
      path_metric = min_node.spf_metric + intf.metric
      if path_metric < intf.remote_node.spf_metric
        intf.remote_node.spf_metric = path_metric
        if min_node is spf_root
          intf.remote_node.next_hops = make_list(intf)
        else
          intf.remote_node.next_hops = min_node.next_hops
      if intf.remote_node.IN_MRT_ISLAND
        intf.remote_node.PATH_HITS_ISLAND = true
      else
        intf.remote_node.PATH_HITS_ISLAND =
          min_node.PATH_HITS_ISLAND
      insert_or_update(spf_heap, intf.remote_node)
    else if path_metric == intf.remote_node.spf_metric
      if min_node is spf_root
        add_to_list(intf.remote_node.next_hops, intf)
      else
        add_list_to_list(intf.remote_node.next_hops,
          min_node.next_hops)
      if intf.remote_node.IN_MRT_ISLAND
        intf.remote_node.PATH_HITS_ISLAND = true
      else
        intf.remote_node.PATH_HITS_ISLAND =
          min_node.PATH_HITS_ISLAND

```

Figure 27: Island_Marking_SPF for determining if an Island Neighbor is loop-free

It is also possible that a given prefix is originated by a combination of non-island routers and island routers. The results of the Island_Marking_SPF computation can be used to determine if the shortest path from an IN to reach that prefix hits the MRT island. The shortest path for the IN to reach prefix P is determined by the total cost to reach prefix P, which is the sum of the cost for the IN to reach a prefix-advertising node and the cost with which that node advertises the prefix. The path with the minimum total cost to prefix P is chosen. If the prefix-advertising node for that minimum total cost path has PATH_HITS_ISLAND set to True, then the IN is not loop-free with respect to the MRT Island for reaching prefix P. If

there multiple minimum total cost paths to reach prefix P, then all of the prefix-advertising routers involved in the minimum total cost paths MUST have `PATH_HITS_ISLAND` set to `False` for the IN to be considered loop-free to reach P.

Note that there are other computations that could be used to determine if paths from a given IN *might* pass through the MRT Island for a given prefix or destination. For example, a previous version of this draft specified running the SPF algorithm on modified topology which treats the MRT island as a single node (with intra-island links set to zero cost) in order to provide input to computations to determine if the path from IN to non-island destination hits the MRT island in this modified topology. This computation is enough to guarantee that a path will not hit the MRT island in the original topology. However, it is possible that a path which is disqualified for hitting the MRT island in the modified topology will not actually hit the MRT Island in the original topology. The algorithm described in `Island_Marking_SPF()` above does not modify the original topology, and will only disqualify a path if the actual path does in fact hit the MRT island.

Since all routers need to come to the same conclusion about which routers qualify as LFINS, this specification requires that all routers computing LFINS MUST use an algorithm whose result is identical to that of the `Island_Marking_SPF()` in Figure 27.

5.9.3. Computing MRT Next-Hops for Proxy-Nodes

Determining the MRT next-hops for a proxy-node in the degenerate case where the proxy-node is attached to only one node in the GADAG is trivial, as all needed information can be derived from that proxy node attachment router. If there are multiple interfaces connecting the proxy node to the single proxy node attachment router, then some can be assigned to MRT-Red and others to MRT-Blue.

Now, consider the proxy-node P that is attached to two proxy-node attachment routers. The pseudo-code for `Select_Proxy_Node_NHs(P,S)` in Figure 28 specifies how a computing-router S MUST compute the MRT red and blue next-hops to reach proxy-node P. The proxy-node attachment router with the lower value of `mrt_node_id` (as defined in Figure 15) is assigned to X, and the other proxy-node attachment router is assigned to Y. We will be using the relative order of X,Y, and S in the partial order defined by the GADAG to determine the MRT red and blue next-hops to reach P, so we also define A and B as the order proxies for X and Y, respectively, with respect to S. The order proxies for all nodes with respect to S were already computed in `Compute_MRT_NextHops()`.

```
def Select_Proxy_Node_NHs(P,S):
    if P.pnar1.node.node_id < P.pnar2.node.node_id:
        X = P.pnar1.node
        Y = P.pnar2.node
    else:
        X = P.pnar2.node
        Y = P.pnar1.node
    P.pnar_X = X
    P.pnar_Y = Y
    A = X.order_proxy
    B = Y.order_proxy
    if (A is S.localroot
        and B is S.localroot):
        // case 1.0
        Copy_List_Items(P.blue_next_hops, X.blue_next_hops)
        Copy_List_Items(P.red_next_hops, Y.red_next_hops)
        return
    if (A is S.localroot
        and B is not S.localroot):
        // case 2.0
        if B.LOWER:
            // case 2.1
            Copy_List_Items(P.blue_next_hops, X.blue_next_hops)
            Copy_List_Items(P.red_next_hops, Y.red_next_hops)
            return
        if B.HIGHER:
            // case 2.2
            Copy_List_Items(P.blue_next_hops, X.red_next_hops)
            Copy_List_Items(P.red_next_hops, Y.blue_next_hops)
            return
        else:
            // case 2.3
            Copy_List_Items(P.blue_next_hops, X.red_next_hops)
            Copy_List_Items(P.red_next_hops, Y.red_next_hops)
            return
    if (A is not S.localroot
        and B is S.localroot):
        // case 3.0
        if A.LOWER:
            // case 3.1
            Copy_List_Items(P.blue_next_hops, X.red_next_hops)
            Copy_List_Items(P.red_next_hops, Y.blue_next_hops)
            return
        if A.HIGHER:
            // case 3.2
            Copy_List_Items(P.blue_next_hops, X.blue_next_hops)
            Copy_List_Items(P.red_next_hops, Y.red_next_hops)
            return
```

```
    else:
        // case 3.3
        Copy_List_Items(P.blue_next_hops, X.red_next_hops)
        Copy_List_Items(P.red_next_hops, Y.red_next_hops)
        return
if (A is not S.localroot
and B is not S.localroot):
    // case 4.0
    if (S is A.localroot or S is B.localroot):
        // case 4.05
        if A.topo_order < B.topo_order:
            // case 4.05.1
            Copy_List_Items(P.blue_next_hops, X.blue_next_hops)
            Copy_List_Items(P.red_next_hops, Y.red_next_hops)
            return
        else:
            // case 4.05.2
            Copy_List_Items(P.blue_next_hops, X.red_next_hops)
            Copy_List_Items(P.red_next_hops, Y.blue_next_hops)
            return
    if A.LOWER:
        // case 4.1
        if B.HIGHER:
            // case 4.1.1
            Copy_List_Items(P.blue_next_hops, X.red_next_hops)
            Copy_List_Items(P.red_next_hops, Y.blue_next_hops)
            return
        if B.LOWER:
            // case 4.1.2
            if A.topo_order < B.topo_order:
                // case 4.1.2.1
                Copy_List_Items(P.blue_next_hops, X.blue_next_hops)
                Copy_List_Items(P.red_next_hops, Y.red_next_hops)
                return
            else:
                // case 4.1.2.2
                Copy_List_Items(P.blue_next_hops, X.red_next_hops)
                Copy_List_Items(P.red_next_hops, Y.blue_next_hops)
                return
        else:
            // case 4.1.3
            Copy_List_Items(P.blue_next_hops, X.red_next_hops)
            Copy_List_Items(P.red_next_hops, Y.red_next_hops)
            return
    if A.HIGHER:
        // case 4.2
        if B.HIGHER:
            // case 4.2.1
```

```
    if A.topo_order < B.topo_order:
        // case 4.2.1.1
        Copy_List_Items(P.blue_next_hops, X.blue_next_hops)
        Copy_List_Items(P.red_next_hops, Y.red_next_hops)
        return
    else:
        // case 4.2.1.2
        Copy_List_Items(P.blue_next_hops, X.red_next_hops)
        Copy_List_Items(P.red_next_hops, Y.blue_next_hops)
        return
    if B.LOWER:
        // case 4.2.2
        Copy_List_Items(P.blue_next_hops, X.blue_next_hops)
        Copy_List_Items(P.red_next_hops, Y.red_next_hops)
        return
    else:
        // case 4.2.3
        Copy_List_Items(P.blue_next_hops, X.blue_next_hops)
        Copy_List_Items(P.red_next_hops, Y.blue_next_hops)
        return
else:
    // case 4.3
    if B.LOWER:
        // case 4.3.1
        Copy_List_Items(P.blue_next_hops, X.red_next_hops)
        Copy_List_Items(P.red_next_hops, Y.red_next_hops)
        return
    if B.HIGHER:
        // case 4.3.2
        Copy_List_Items(P.blue_next_hops, X.blue_next_hops)
        Copy_List_Items(P.red_next_hops, Y.blue_next_hops)
        return
    else:
        // case 4.3.3
        if A.topo_order < B.topo_order:
            // case 4.3.3.1
            Copy_List_Items(P.blue_next_hops, X.blue_next_hops)
            Copy_List_Items(P.red_next_hops, Y.red_next_hops)
            return
        else:
            // case 4.3.3.2
            Copy_List_Items(P.blue_next_hops, X.red_next_hops)
            Copy_List_Items(P.red_next_hops, Y.blue_next_hops)
            return
assert(False)
```

Figure 28: Select_Proxy_Node_NHs()

It is useful to understand up front that the blue next-hops to reach proxy-node P produced by `Select_Proxy_Node_NHs()` will always be the next-hops that reach proxy-node attachment router X, while the red next-hops to reach proxy-node P will always be the next-hops that reach proxy-node attachment router Y. This is different from the red and blue next-hops produced by `Compute_MRT_NextHops()` where, for example, blue next-hops to a destination that is ordered with respect to the source will always correspond to an INCREASING next-hop on the GADAG. The exact choice of which next-hops chosen by `Select_Proxy_Node_NHs()` as the blue next-hops to reach P (which will necessarily go through X on its way to P) does depend on the GADAG, but the relationship is more complex than was the case with `Compute_MRT_NextHops()`.

There are twenty-one different relative order relationships between A, B and S that `Select_Proxy_Node_NHs()` uses to determine red and blue next-hops to P. This document does not attempt to provide an exhaustive description of each case considered in `Select_Proxy_Node_NHs()`. Instead we provide a high level overview of the different cases, and we consider a few cases in detail to give an example of the reasoning that can be used to understand each case.

At the highest level, `Select_Proxy_Node_NHs()` distinguishes between four different cases depending on whether or not A or B is the localroot for S. For example, for case 4.0, neither A nor B is the localroot for S. Case 4.05 addresses the case where S is the localroot for either A or B, while cases 4.1, 4.2, and 4.3 address the cases where A is ordered lower than S, A is ordered higher than S, or A is unordered with respect to S on the GADAG. In general, each of these cases is then further subdivided into whether or not B is ordered lower than S, B is ordered higher than S, or B is unordered with respect to S. In some cases we also need a further level of discrimination, where we use the topological sort order of A with respect to B.

As a detailed example, let's consider case 4.1 and all of its sub-cases, and explain why the red and blue next-hops to reach P are chosen as they are in `Select_Proxy_Node_NHs()`. In case 4.1, neither A nor B is the localroot for S, S is not the localroot for A or B, and A is ordered lower than S on the GADAG. In this situation, we know that the red path to reach X (as computed in `Compute_MRT_NextHops()`) will follow DECREASING next-hops towards A, while the blue path to reach X will follow INCREASING next-hops to the localroot, and then INCREASING next-hops to A.

Now consider sub-case 4.1.1 where B is ordered higher than S. In this situation, we know that the blue path to reach Y will follow INCREASING next-hops towards B, while the red next-hops to reach Y

will follow DECREASING next-hops to the localroot, and then DECREASING next-hops to B. So to reach X and Y by two disjoint paths, we can choose the red next-hops to X and the blue next-hops to Y. We have chosen the convention that blue next-hops to P are those that pass through X, and red next-hops to P are those that pass through Y, so we can see that case 4.1.1 produces the desired result. Choosing blue to X and red to Y does not produce disjoint paths because the paths intersect at least at the localroot.

Now consider sub-case 4.1.2 where B is ordered lower than S. In this situation, we know that the red path to reach Y will follow DECREASING next-hops towards B, while the BLUE next-hops to reach Y will follow INCREASING next-hops to the localroot, and then INCREASING next-hops to A. The choice here is more difficult than in 4.1.1 because A and B are both on the DECREASING path from S towards the localroot. We want to use the direct DECREASING(red) path to the one that is nearer to S on the GADAG. We get this extra information by comparing the topological sort order of A and B. If $A.topo_order < B.topo_order$, then we use red to Y and blue to X, since the red path to Y will DECREASE to B without hitting A, and the blue path to X will INCREASE to A without hitting B. Instead, if $A.topo_order > B.topo_order$, then we use red to X and blue to Y.

Note that when A is unordered with respect to B, the result of comparing $A.topo_order$ with $B.topo_order$ could be greater than or less than. In this case, the result doesn't matter because either choice (red to Y and blue to X or red to X and blue to Y) would work. What is required is that all nodes in the network give the same result when comparing $A.topo_order$ with $B.topo_order$. This is guaranteed by having all nodes run the same algorithm (`Run_Topological_Sort_GADAG()`) to compute the topological sort order.

Finally we consider case 4.1.3, where B is unordered with respect to S. In this case, the blue path to reach Y will follow the DECREASING next-hops towards the localroot until it reaches some node (K) which is ordered less than B, after which it will take INCREASING next-hops to B. The red path to reach Y will follow the INCREASING next-hops towards the localroot until it reaches some node (L) which is ordered greater than B, after which it will take DECREASING next-hops to B. Both K and A are reached by DECREASING from S, but we don't have information about whether or not that DECREASING path will hit K or A first. Instead, we do know that the INCREASING path from S will hit L before reaching A. Therefore, we use the red path to reach Y and the red path to reach X.

Similar reasoning can be applied to understand the other seventeen cases used in `Select_Proxy_Node_NHs()`. However, cases 2.3 and 3.3 deserve special attention because the correctness of the solution for

these two cases relies on a special property of the GADAGs that we have constructed in this algorithm, a property not shared by all GADAGs in general. Focusing on case 2.3, we consider the case where A is the localroot for S, while B is not, and B is unordered with respect to S. The red path to X DECREASES from S to the localroot A, while the blue path to X INCREASES from S to the localroot A. The blue path to Y DECREASES towards the localroot A until it reaches some node (K) which is ordered less than B, after which the path INCREASES to B. The red path to Y INCREASES towards the localroot A until it reaches some node (L) which is ordered greater than B, after which the path DECREASES to B. It can be shown that for an arbitrary GADAG, with only the ordering relationships computed so far, we don't have enough information to choose a pair of paths to reach X and Y that are guaranteed to be disjoint. In some topologies, A will play the role of K, the first node ordered less than B on the blue path to Y. In other topologies, A will play the role of L, the first node ordered greater than B on the red path to Y. The basic problem is that we cannot distinguish between these two cases based on the ordering relationships.

As discussed Section 5.8, the GADAGs that we construct using the algorithm in this document are not arbitrary GADAGs. They have the additional property that incoming links to a localroot come from only one other node in the same block. This is a result of the method of construction. This additional property guarantees that localroot A will never play the role of L in the red path to Y, since L must have at least two incoming links from different nodes in the same block in the GADAG. This in turn allows `Select_Proxy_Node_NHs()` to choose the red path to Y and the red path to X as the disjoint MRT paths to reach P.

5.9.4. Computing MRT Alternates for Proxy-Nodes

After finding the red and the blue next-hops for a given proxy-node P, it is necessary to know which one of these to use in the case of failure. This can be done by `Select_Alternates_Proxy_Node()`, as shown in the pseudo-code in Figure 29.

```
def Select_Alternates_Proxy_Node(P,F,primary_intf):
    S = primary_intf.local_node
    X = P.pnar_X
    Y = P.pnar_Y
    A = X.order_proxy
    B = Y.order_proxy
    if F is A and F is B:
        return 'PRIM_NH_IS_OP_FOR_BOTH_X_AND_Y'
    if F is A:
        return 'USE_RED'
```

```
if F is B:
    return 'USE_BLUE'

if not In_Common_Block(A, B):
    if In_Common_Block(F, A):
        return 'USE_RED'
    elif In_Common_Block(F, B):
        return 'USE_BLUE'
    else:
        return 'USE_RED_OR_BLUE'
if (not In_Common_Block(F, A)
    and not In_Common_Block(F, B) ):
    return 'USE_RED_OR_BLUE'

alt_to_X = Select_Alternates(X, F, primary_intf)
alt_to_Y = Select_Alternates(Y, F, primary_intf)

if (alt_to_X == 'USE_RED_OR_BLUE'
    and alt_to_Y == 'USE_RED_OR_BLUE'):
    return 'USE_RED_OR_BLUE'
if alt_to_X == 'USE_RED_OR_BLUE':
    return 'USE_BLUE'
if alt_to_Y == 'USE_RED_OR_BLUE':
    return 'USE_RED'

if (A is S.localroot
    and B is S.localroot):
    // case 1.0
    if (alt_to_X == 'USE_BLUE' and alt_to_Y == 'USE_RED'):
        return 'USE_RED_OR_BLUE'
    if alt_to_X == 'USE_BLUE':
        return 'USE_BLUE'
    if alt_to_Y == 'USE_RED':
        return 'USE_RED'
    assert(False)
if (A is S.localroot
    and B is not S.localroot):
    // case 2.0
    if B.LOWER:
        // case 2.1
        if (alt_to_X == 'USE_BLUE' and alt_to_Y == 'USE_RED'):
            return 'USE_RED_OR_BLUE'
        if alt_to_X == 'USE_BLUE':
            return 'USE_BLUE'
        if alt_to_Y == 'USE_RED':
            return 'USE_RED'
        assert(False)
    if B.HIGHER:
```

```
        // case 2.2
        if (alt_to_X == 'USE_RED' and alt_to_Y == 'USE_BLUE'):
            return 'USE_RED_OR_BLUE'
        if alt_to_X == 'USE_RED':
            return 'USE_BLUE'
        if alt_to_Y == 'USE_BLUE':
            return 'USE_RED'
        assert(False)
    else:
        // case 2.3
        if (alt_to_X == 'USE_RED' and alt_to_Y == 'USE_RED'):
            return 'USE_RED_OR_BLUE'
        if alt_to_X == 'USE_RED':
            return 'USE_BLUE'
        if alt_to_Y == 'USE_RED':
            return 'USE_RED'
        assert(False)
if (A is not S.localroot
and B is S.localroot):
    // case 3.0
    if A.LOWER:
        // case 3.1
        if (alt_to_X == 'USE_RED' and alt_to_Y == 'USE_BLUE'):
            return 'USE_RED_OR_BLUE'
        if alt_to_X == 'USE_RED':
            return 'USE_BLUE'
        if alt_to_Y == 'USE_BLUE':
            return 'USE_RED'
        assert(False)
    if A.HIGHER:
        // case 3.2
        if (alt_to_X == 'USE_BLUE' and alt_to_Y == 'USE_RED'):
            return 'USE_RED_OR_BLUE'
        if alt_to_X == 'USE_BLUE':
            return 'USE_BLUE'
        if alt_to_Y == 'USE_RED':
            return 'USE_RED'
        assert(False)
    else:
        // case 3.3
        if (alt_to_X == 'USE_RED' and alt_to_Y == 'USE_RED'):
            return 'USE_RED_OR_BLUE'
        if alt_to_X == 'USE_RED':
            return 'USE_BLUE'
        if alt_to_Y == 'USE_RED':
            return 'USE_RED'
        assert(False)
if (A is not S.localroot
```

```
and B is not S.localroot):
// case 4.0
if (S is A.localroot or S is B.localroot):
// case 4.05
if A.topo_order < B.topo_order:
// case 4.05.1
if (alt_to_X == 'USE_BLUE' and alt_to_Y == 'USE_RED'):
return 'USE_RED_OR_BLUE'
if alt_to_X == 'USE_BLUE':
return 'USE_BLUE'
if alt_to_Y == 'USE_RED':
return 'USE_RED'
assert(False)
else:
// case 4.05.2
if (alt_to_X == 'USE_RED' and alt_to_Y == 'USE_BLUE'):
return 'USE_RED_OR_BLUE'
if alt_to_X == 'USE_RED':
return 'USE_BLUE'
if alt_to_Y == 'USE_BLUE':
return 'USE_RED'
assert(False)
if A.LOWER:
// case 4.1
if B.HIGHER:
// case 4.1.1
if (alt_to_X == 'USE_RED' and alt_to_Y == 'USE_BLUE'):
return 'USE_RED_OR_BLUE'
if alt_to_X == 'USE_RED':
return 'USE_BLUE'
if alt_to_Y == 'USE_BLUE':
return 'USE_RED'
assert(False)
if B.LOWER:
// case 4.1.2
if A.topo_order < B.topo_order:
// case 4.1.2.1
if (alt_to_X == 'USE_BLUE'
and alt_to_Y == 'USE_RED'):
return 'USE_RED_OR_BLUE'
if alt_to_X == 'USE_BLUE':
return 'USE_BLUE'
if alt_to_Y == 'USE_RED':
return 'USE_RED'
assert(False)
else:
// case 4.1.2.2
if (alt_to_X == 'USE_RED'
```

```
        and alt_to_Y == 'USE_BLUE'):
            return 'USE_RED_OR_BLUE'
    if alt_to_X == 'USE_RED':
        return 'USE_BLUE'
    if alt_to_Y == 'USE_BLUE':
        return 'USE_RED'
    assert(False)
else:
    // case 4.1.3
    if (F.LOWER and not F.HIGHER
        and F.topo_order > A.topo_order):
        // case 4.1.3.1
        return 'USE_RED'
    else:
        // case 4.1.3.2
        return 'USE_BLUE'
if A.HIGHER:
    // case 4.2
    if B.HIGHER:
        // case 4.2.1
        if A.topo_order < B.topo_order:
            // case 4.2.1.1
            if (alt_to_X == 'USE_BLUE'
                and alt_to_Y == 'USE_RED'):
                return 'USE_RED_OR_BLUE'
            if alt_to_X == 'USE_BLUE':
                return 'USE_BLUE'
            if alt_to_Y == 'USE_RED':
                return 'USE_RED'
            assert(False)
        else:
            // case 4.2.1.2
            if (alt_to_X == 'USE_RED'
                and alt_to_Y == 'USE_BLUE'):
                return 'USE_RED_OR_BLUE'
            if alt_to_X == 'USE_RED':
                return 'USE_BLUE'
            if alt_to_Y == 'USE_BLUE':
                return 'USE_RED'
            assert(False)
    if B.LOWER:
        // case 4.2.2
        if (alt_to_X == 'USE_BLUE'
            and alt_to_Y == 'USE_RED'):
            return 'USE_RED_OR_BLUE'
        if alt_to_X == 'USE_BLUE':
            return 'USE_BLUE'
        if alt_to_Y == 'USE_RED':
```

```
        return 'USE_RED'
    assert(False)
else:
    // case 4.2.3
    if (F.HIGHER and not F.LOWER
        and F.topo_order < A.topo_order):
        return 'USE_RED'
    else:
        return 'USE_BLUE'
else:
    // case 4.3
    if B.LOWER:
        // case 4.3.1
        if (F.LOWER and not F.HIGHER
            and F.topo_order > B.topo_order):
            return 'USE_BLUE'
        else:
            return 'USE_RED'
    if B.HIGHER:
        // case 4.3.2
        if (F.HIGHER and not F.LOWER
            and F.topo_order < B.topo_order):
            return 'USE_BLUE'
        else:
            return 'USE_RED'
    else:
        // case 4.3.3
        if A.topo_order < B.topo_order:
            // case 4.3.3.1
            if (alt_to_X == 'USE_BLUE'
                and alt_to_Y == 'USE_RED'):
                return 'USE_RED_OR_BLUE'
            if alt_to_X == 'USE_BLUE':
                return 'USE_BLUE'
            if alt_to_Y == 'USE_RED':
                return 'USE_RED'
            assert(False)
        else:
            // case 4.3.3.2
            if (alt_to_X == 'USE_RED'
                and alt_to_Y == 'USE_BLUE'):
                return 'USE_RED_OR_BLUE'
            if alt_to_X == 'USE_RED':
                return 'USE_BLUE'
            if alt_to_Y == 'USE_BLUE':
                return 'USE_RED'
            assert(False)
assert(False)
```

Figure 29: `Select_Alternates_Proxy_Node()`

`Select_Alternates_Proxy_Node(P,F,primary_intf)` determines whether it is safe to use the blue path to P (which goes through X), the red path to P (which goes through Y), or either, when the `primary_intf` to node F (and possibly node F) fails. The basic approach is to run `Select_Alternates(X,F,primary_intf)` and `Select_Alternates(Y,F,primary_intf)` to determine which of the two MRT paths to X and which of the two MRT paths to Y is safe to use in the event of the failure of F. In general, we will find that if it is safe to use a particular path to X or Y when F fails, and `Select_Proxy_Node_NHs()` used that path when constructing the red or blue path to reach P, then it will also be safe to use that path to reach P when F fails. This rule has one exception which is covered below. First, we give a concrete example of how `Select_Alternates_Proxy_Node()` works in the common case.

The twenty one ordering relationships used in `Select_Proxy_Node_NHs()` are repeated in `Select_Alternates_Proxy_Node()`. We focus on case 4.1.1 to give a detailed example of the reasoning used in `Select_Alternates_Proxy_Node()`. In `Select_Proxy_Node_NHs()`, we determined for case 4.1.1 that the red next-hops to X and the blue next-hops to Y allow us to reach X and Y by disjoint paths, and are thus the blue and red next-hops to reach P. Therefore, if we run `Select_Alternates(X, F, primary_intf)` and we find that it is safe to USE_RED to reach X, then we also conclude that it is safe to use the MRT path through X to reach P (the blue path to P) when F fails. Similarly, if run `Select_Alternates(X, F, primary_intf)` and we find that it is safe to USE_BLUE to reach Y, then we also conclude that it is safe to use the MRT path through Y to reach P (the red path to P) when F fails. If both of the paths that were used in `Select_Proxy_Node_NHs()` to construct the blue and red paths to P are found to be safe to use to reach X and Y, then we conclude that we can use either the red or the blue path to P.

This simple reasoning gives the correct answer in most of the cases. However, additional logic is needed when either A or B (but not both A and B) is unordered with respect to S. This applies to cases 4.1.3, 4.2.3, 4.3.1, and 4.3.2. Looking at case 4.1.3 in more detail, A is ordered less than S, but B is unordered with respect to S. In the discussion of case 4.1.3 above, we saw that `Select_Proxy_Node_NHs()` chose the red path to reach Y and the red path to reach X. We also saw that the red path to reach Y will follow the INCREASING next-hops towards the localroot until it reaches some node (L) which is ordered greater than B, after which it will take DECREASING next-hops to B. The problem is that the red path to reach P (the one that goes through Y) won't necessarily be the same as the red path to reach Y. This is because the next-hop

that node L computes for its red next-hop to reach P may be different from the next-hop it computes for its red next-hop to reach Y. This is because B is ordered lower than L, so L applies case 4.1.2 of `Select_Proxy_Node_NHs()` in order to determine its next-hops to reach P. If $A.topo_order < B.topo_order$ (case 4.1.2.1), then L will choose DECREASING next-hops directly to B, which is the same result that L computes in `Compute_MRT_NextHops()` to reach Y. However, if $A.topo_order > B.topo_order$ (case 4.1.2.2), then L will choose INCREASING next-hops to reach B, which is different from what L computes in `Compute_MRT_NextHops()` to reach Y. So testing the safety of the path for S to reach Y on failure of F as a surrogate for the safety of using the red path to reach P is not reliable in this case. It is possible to construct topologies where the red path to P hits F even though the red path to Y does not hit F.

Fortunately there is enough information in the order relationships that we have already computed to still figure out which alternate to choose in these four cases. The basic idea is to always choose the path involving the ordered node, unless that path would hit F. Returning to case 4.1.3, we see that since A is ordered lower than S, the only way for S to hit F using a simple DECREASING path to A is for F to lie between A and S on the GADAG. This scenario is covered by requiring that F be lower than S (but not also higher than S) and that $F.topo_order > A.topo_order$ in case 4.1.3.1.

We just need to confirm that it is safe to use the path involving B in this scenario. In case 4.1.3.1, either F is between A and S on the GADAG, or F is unordered with respect to A and lies on the DECREASING path from S to the localroot. When F is between A and S on the GADAG, then the path through B chosen to avoid A in `Select_Proxy_Node_NHs()` will also avoid F. When F is unordered with respect to A and lies on the DECREASING path from S to the localroot, then we consider two cases. Either $F.topo_order < B.topo_order$ or $F.topo_order > B.topo_order$. In the first case, since $F.topo_order < B.topo_order$ and $F.topo_order > A.topo_order$, it must be the case that $A.topo_order < B.topo_order$. Therefore, L will choose DECREASING next-hops directly to B (case 4.1.2.1), which cannot hit F since $F.topo_order < B.topo_order$. In the second case, where $F.topo_order > B.topo_order$, the only way for the path involving B to hit F is if it DECREASES from L to B through F, ie. it must be that $L >> F >> B$. However, since $S >> F$, this would imply that $S >> B$. However, we know that S is unordered with respect to B, so the second case cannot occur. So we have demonstrated that the red path to P (which goes via B and Y) is safe to use under the conditions of 4.1.3.1. Similar reasoning can be applied to the other three special cases where either A or B is unordered with respect to S.

6. MRT Lowpoint Algorithm: Next-hop conformance

This specification defines the MRT Lowpoint Algorithm, which include the construction of a common GADAG and the computation of MRT-Red and MRT-Blue next-hops to each node in the graph. An implementation MAY select any subset of next-hops for MRT-Red and MRT-Blue that respect the available nodes that are described in Section 5.7 for each of the MRT-Red and MRT-Blue and the selected next-hops are further along in the interval of allowed nodes towards the destination.

For example, the MRT-Blue next-hops used when the destination $Y \gg X$, the computing router, MUST be one or more nodes, T, whose `topo_order` is in the interval $[X.topo_order, Y.topo_order]$ and where $Y \gg T$ or Y is T. Similarly, the MRT-Red next-hops MUST be have a `topo_order` in the interval $[R-small.topo_order, X.topo_order]$ or $[Y.topo_order, R-big.topo_order]$.

Implementations SHOULD implement the `Select_Alternates()` function to pick an MRT-FRR alternate.

7. Broadcast interfaces

When broadcast interfaces are used to connect nodes, the broadcast network MUST be represented as a pseudonode, where each real node connects to the pseudonode. The interface metric in the direction from real node to pseudonode is the non-zero interface metric, while the interface metric in the direction from the pseudonode to the real node is set to zero. This is consistent with the way that broadcast interfaces are represented as pseudonodes in IS-IS and OSPF.

Pseudonodes MUST be treated as equivalent to real nodes in the network graph used in the MRT algorithm with a few exceptions detailed below.

The pseudonodes MUST be included in the computation of the GADAG. The neighbors of the pseudonode need to know the `mrt_node_id` of the pseudonode in order to consistently order interfaces, which is needed to compute the GADAG. The `mrt_node_id` for IS-IS is the 7 octet neighbor system ID and pseudonode number in TLV #22 or TLV#222. The `mrt_node_id` for OSPFv2 is the 4 octet interface address of the Designated Router found in the Link ID field for the link type 2 (transit network) in the Router-LSA. The `mrt_node_id` for OSPFv3 is the 4 octet interface address of the Designated Router found in the Neighbor Interface ID field for the link type 2 (transit network) in the Router-LSA. pseudonodes MUST NOT be considered as candidates for GADAG root selection. Note that this is different from the Neighbor Router ID field used for the `mrt_node_id` for point-to-point links in OSPFv3 Router-LSAs given in Figure 15.

Pseudonodes MUST NOT be considered as candidates for selection as GADAG root. This rule is intended to result in a more stable network-wide selection of GADAG root by removing the possibility that the change of Designated Router or Designated Intermediate System on a broadcast network can result in a change of GADAG root.

7.1. Computing MRT next-hops on broadcast networks

The pseudonode does not correspond to a real node, so it is not actually involved in forwarding. A real node on a broadcast network cannot simply forward traffic to the broadcast network. It must specify another real node on the broadcast network as the next-hop. On a network graph where a broadcast network is represented by a pseudonode, this means that if a real node determines that the next-hop to reach a given destination is a pseudonode, it must also determine the next-next-hop for that destination in the network graph, which corresponds to a real node attached to the broadcast network.

It is interesting to note that this issue is not unique to the MRT algorithm, but is also encountered in normal SPF computations for IGP. Section 16.1.1 of [RFC2328] describes how this is done for OSPF. As OSPF runs Dijkstra's algorithm, whenever a shorter path is found reach a real destination node, and the shorter path is one hop from the computing routing, and that one hop is a pseudonode, then the next-hop for that destination is taken from the interface IP address in the Router-LSA correspond to the link to the real destination node

For IS-IS, in the example pseudo-code implementation of Dijkstra's algorithm in Annex C of [ISO10589-Second-Edition] whenever the algorithm encounters an adjacency from a real node to a pseudonode, it gets converted to a set of adjacencies from the real node to the neighbors of the pseudonode. In this way, the computed next-hops point all the way to the real node, and not the pseudonode.

We could avoid the problem of determining next-hops across pseudonodes in MRT by converting the pseudonode representation of broadcast networks to a full mesh of links between real nodes on the same network. However, if we make that conversion before computing the GADAG, we lose information about which links actually correspond to a single physical interface into the broadcast network. This could result computing red and blue next-hops that use the same broadcast interface, in which case neither the red nor the blue next-hop would be usable as an alternate on failure of the broadcast interface.

Instead, we take the following approach, which maintains the property that either the red and blue next-hop will avoid the broadcast network, if topologically allowed. We run the MRT algorithm treating the pseudonodes as equivalent to real nodes in the network graph, with the exceptions noted above. In addition to running the MRT algorithm from the point of view of itself, a computing router connected to a pseudonode MUST also run the MRT algorithm from the point of view of each of its pseudonode neighbors. For example, if a computing router S determines that its MRT red next-hop to reach a destination D is a pseudonode P, S looks at its MRT algorithm computation from P's point of view to determine P's red next-hop to reach D, say interface 1 on node X. S now knows that its real red next-hop to reach D is interface 1 on node X on the broadcast network represented by P, and can install the corresponding entry in its FIB.

7.2. Using MRT next-hops as alternates in the event of failures on broadcast networks

In the previous section, we specified how to compute MRT next-hops when broadcast networks are involved. In this section, we discuss how a PLR can use those MRT next-hops in the event of failures involving broadcast networks.

A PLR attached to a broadcast network running only OSPF or IS-IS with large Hello intervals has limited ability to quickly detect failures on a broadcast network. The only failure mode that can be quickly detected is the failure of the physical interface connecting the PLR to the broadcast network. For the failure of the interface connecting the PLR to the broadcast network, the alternate that avoids the broadcast network can be computed by using the broadcast network pseudonode as F, the primary next-hop node, in `Select_Alternates()`. This will choose an alternate path that avoids the broadcast network. However, the alternate path will not necessarily avoid all of the real nodes connected to the broadcast network. This is because we have used the pseudonode to represent the broadcast network. And we have enforced the node-protecting property of MRT on the pseudonode to provide protection against failure of the broadcast network, not the real next-hop nodes on the broadcast network. This is the best that we can hope to do if failure of the broadcast interface is the only failure mode that the PLR can respond to.

We can improve on this if the PLR also has the ability to quickly detect a lack of connectivity across the broadcast network to a given IP-layer node. This can be accomplished by running BFD between all pairs of IGP neighbors on the broadcast network. Note that in the case of OSPF, this would require establishing BFD sessions between all pairs of neighbors in the 2-WAY state. When the PLR can quickly

detect the failure of a particular next-hop across a broadcast network, then the PLR can be more selective in its choice of alternates. For example, when the PLR observes that connectivity to an IP-layer node on a broadcast network has failed, the PLR may choose to still use the broadcast network to reach other IP-layer nodes which are still reachable. Or if the PLR observes that connectivity has failed to several IP-layer nodes on the same broadcast network, it may choose to treat the entire broadcast network as failed. The choice of MRT alternates by a PLR for a particular set of failure conditions is a local decision, since it does not require coordination with other nodes.

8. Evaluation of Alternative Methods for Constructing GADAGs

This document specifies the MRT Lowpoint algorithm. One component of the algorithm involves constructing a common GADAG based on the network topology. The MRT Lowpoint algorithm computes the GADAG using the method described in Section 5.5. This method aims to minimize the amount of computation required to compute the GADAG. In the process of developing the MRT Lowpoint algorithm, two alternative methods for constructing GADAGs were also considered. These alternative methods are described in Appendix B and Appendix C. In general, these other two methods require more computation to compute the GADAG. The analysis below was performed to determine if the alternative GADAG construction methods produce shorter MRT alternate paths in real network topologies, and if so, to what extent.

Figure 30 compares results obtained using the three different methods for constructing GADAGs on five different service provider network topologies. MRT_LOWPOINT indicates the method specified in Section 5.5, while MRT_SPF and MRT_HYBRID indicate the methods specified in Appendix B and Appendix C, respectively. The columns on the right present the distribution of alternate path lengths for each GADAG construction method. Each MRT computation was performed using a same GADAG root chosen based on centrality.

For three of the topologies analyzed (T201, T206, and T211), the use of MRT_SPF or MRT_HYBRID methods does not appear to provide a significantly shorter alternate path lengths compared to the MRT_LOWPOINT method. However, for two of the topologies (T216 and T219), the use of the MRT_SPF method resulted in noticeably shorter alternate path lengths than the use of the MRT_LOWPOINT or MRT_HYBRID methods.

It was decided to use the MRT_LOWPOINT method to construct the GADAG in the algorithm specified in this draft, in order to initially offer an algorithm with lower computational requirements. These results indicate that in the future it may be useful to evaluate and

potentially specify other MRT algorithm variants that use different GADAG construction methods.

Topology name GADAG construction method	percentage of failure scenarios protected by an alternate N hops longer than the primary path								
	0-1	2-3	4-5	6-7	8-9	10	12	14	no alt <16

T201(avg primary hops=3.5)									
MRT_HYBRID	33	26	23	6	3				
MRT_SPF	33	36	23	6	3				
MRT_LOWPOINT	33	36	23	6	3				

T206(avg primary hops=3.7)									
MRT_HYBRID	50	35	13	2					
MRT_SPF	50	35	13	2					
MRT_LOWPOINT	55	32	13						

T211(avg primary hops=3.3)									
MRT_HYBRID	86	14							
MRT_SPF	86	14							
MRT_LOWPOINT	85	15	1						

T216(avg primary hops=5.2)									
MRT_HYBRID	23	22	18	13	10	7	4	2	2
MRT_SPF	35	32	19	9	3	1			
MRT_LOWPOINT	28	25	18	11	7	6	3	2	1

T219(avg primary hops=7.7)									
MRT_HYBRID	20	16	13	10	7	5	5	5	3
MRT_SPF	31	23	19	12	7	4	2	1	
MRT_LOWPOINT	19	14	15	12	10	8	7	6	10

Figure 30

9. Implementation Status

[RFC Editor: please remove this section prior to publication.]

Please see [I-D.ietf-rtgwg-mrt-frr-architecture] for details on implementation status.

10. Operational Considerations

This sections discusses operational considerations related to the the MRT Lowpoint algorithm and other potential MRT algorithm variants. For a discussion of operational considerations related to MRT-FRR in general, see the Operational Considerations section of [I-D.ietf-rtgwg-mrt-frr-architecture].

10.1. GADAG Root Selection

The Default MRT Profile uses the GADAG Root Selection Priority advertised by routers as the primary criterion for selecting the GADAG root. It is RECOMMENDED that an operator designate a set of routers as good choices for selection as GADAG root by setting the GADAG Root Selection Priority for that set of routers to lower (more preferred) numerical values. Criteria for making this designation are discussed below.

Analysis has shown that the centrality of a router can have a significant impact on the lengths of the alternate paths computed. Therefore, it is RECOMMENDED that off-line analysis that considers the centrality of a router be used to help determine how good a choice a particular router is for the role of GADAG root.

If the router currently selected as GADAG root becomes unreachable in the IGP topology, then a new GADAG root will be selected. Changing the GADAG root can change the overall structure of the GADAG as well the paths of the red and blue MRT trees built using that GADAG. In order to minimize change in the associated red and blue MRT forwarding entries that can result from changing the GADAG root, it is RECOMMENDED that operators prioritize for selection as GADAG root those routers that are expected to consistently remain part of the IGP topology.

10.2. Destination-rooted GADAGs

The MRT Lowpoint algorithm constructs a single GADAG rooted at a single node selected as the GADAG root. It is also possible to construct a different GADAG for each destination, with the GADAG rooted at the destination. A router can compute the MRT-Red and MRT-Blue next-hops for that destination based on the GADAG rooted at that destination. Building a different GADAG for each destination is computationally more expensive, but it may give somewhat shorter alternate paths. Using destination-rooted GADAGs would require a new MRT profile to be created with a new MRT algorithm specification, since all routers in the MRT Island would need to use destination-rooted GADAGs.

11. Acknowledgements

The authors would like to thank Shraddha Hegde, Eric Wu, Janos Farkas, Stewart Bryant, and Alvaro Retana for their suggestions and review. We would also like to thank Anil Kumar SN for his assistance in clarifying the algorithm description and pseudo-code.

12. IANA Considerations

This document includes no request to IANA.

13. Security Considerations

The algorithm described in this document does not introduce new security concerns beyond those already discussed in the document describing the MRT FRR architecture [I-D.ietf-rtgwg-mrt-frr-architecture].

14. References

14.1. Normative References

- [I-D.ietf-rtgwg-mrt-frr-architecture]
Atlas, A., Kebler, R., Bowers, C., Envedi, G., Csaszar, A., Tantsura, J., and R. White, "An Architecture for IP/LDP Fast-Reroute Using Maximally Redundant Trees", draft-ietf-rtgwg-mrt-frr-architecture-07 (work in progress), October 2015.
- [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>>.

14.2. Informative References

- [EnyediThesis]
Enyedi, G., "Novel Algorithms for IP Fast Reroute", Department of Telecommunications and Media Informatics, Budapest University of Technology and Economics Ph.D. Thesis, February 2011, <http://www.omikk.bme.hu/collection/s/phd/Villamosmernoki_es_Informatikai_Kar/2011/Enyedi_Gabor/ertekezes.pdf>.

- [IEEE8021Qca]
IEEE 802.1, "IEEE 802.1Qca Bridges and Bridged Networks - Amendment: Path Control and Reservation - Draft 2.1", (work in progress), June 24, 2015, <<http://www.ieee802.org/1/pages/802.1ca.html>>.
- [ISO10589-Second-Edition]
International Organization for Standardization, "Intermediate system to Intermediate system intra-domain routing 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.
- [Kahn_1962_topo_sort]
Kahn, A., "Topological sorting of large networks", Communications of the ACM, Volume 5, Issue 11, Nov 1962, <<http://dl.acm.org/citation.cfm?doid=368996.369025>>.
- [MRTLlinear]
Enyedi, G., Retvari, G., and A. Csaszar, "On Finding Maximally Redundant Trees in Strictly Linear Time", IEEE Symposium on Computers and Communications (ISCC), 2009, <<http://opti.tmit.bme.hu/~enyedi/ipfrr/distMaxRedTree.pdf>>.
- [RFC2328] Moy, J., "OSPF Version 2", STD 54, RFC 2328, DOI 10.17487/RFC2328, April 1998, <<http://www.rfc-editor.org/info/rfc2328>>.
- [RFC5120] Przygienda, T., Shen, N., and N. Sheth, "M-ISIS: Multi Topology (MT) Routing in Intermediate System to Intermediate Systems (IS-ISs)", RFC 5120, DOI 10.17487/RFC5120, February 2008, <<http://www.rfc-editor.org/info/rfc5120>>.
- [RFC7490] Bryant, S., Filsfils, C., Previdi, S., Shand, M., and N. So, "Remote Loop-Free Alternate (LFA) Fast Reroute (FRR)", RFC 7490, DOI 10.17487/RFC7490, April 2015, <<http://www.rfc-editor.org/info/rfc7490>>.

Appendix A. Python Implementation of MRT Lowpoint Algorithm

Below is Python code implementing the MRT Lowpoint algorithm specified in this document. In order to avoid the page breaks in the .txt version of the draft, one can cut and paste the Python code from the .xml version. The code is also posted on Github.

While this Python code is believed to correctly implement the pseudo-code description of the algorithm, in the event of a difference, the pseudo-code description should be considered normative.

<CODE BEGINS>

```
# This program has been tested to run on Python 2.6 and 2.7
# (specifically Python 2.6.6 and 2.7.8 were tested).
# The program has known incompatibilities with Python 3.X.

# When executed, this program will generate a text file describing
# an example topology. It then reads that text file back in as input
# to create the example topology, and runs the MRT algorithm. This
# was done to simplify the inclusion of the program as a single text
# file that can be extracted from the IETF draft.

# The output of the program is four text files containing a description
# of the GADAG, the blue and red MRTs for all destinations, and the
# MRT alternates for all failures.
```

```
import random
import os.path
import heapq
```

```
# simple Class definitions allow structure-like dot notation for
# variables and a convenient place to initialize those variables.
```

```
class Topology:
```

```
    def __init__(self):
        self.gadag_root = None
        self.node_list = []
        self.node_dict = {}
        self.test_gr = None
        self.island_node_list_for_test_gr = []
        self.stored_named_proxy_dict = {}
        self.init_new_computing_router()
    def init_new_computing_router(self):
        self.island_node_list = []
        self.named_proxy_dict = {}
```

```
class Node:
```

```
    def __init__(self):
        self.node_id = None
        self.intf_list = []
        self.profile_id_list = [0]
        self.GR_sel_priority = 128
        self.blue_next_hops_dict = {}
        self.red_next_hops_dict = {}
        self.blue_to_green_nh_dict = {}
        self.red_to_green_nh_dict = {}
```

```
        self.prefix_cost_dict = {}
        self.pnh_dict = {}
        self.alt_dict = {}
        self.init_new_computing_router()
def init_new_computing_router(self):
    self.island_intf_list = []
    self.IN_MRT_ISLAND = False
    self.IN_GADAG = False
    self.dfs_number = None
    self.dfs_parent = None
    self.dfs_parent_intf = None
    self.dfs_child_list = []
    self.lowpoint_number = None
    self.lowpoint_parent = None
    self.lowpoint_parent_intf = None
    self.localroot = None
    self.block_id = None
    self.IS_CUT_VERTEX = False
    self.blue_next_hops = []
    self.red_next_hops = []
    self.primary_next_hops = []
    self.alt_list = []

class Interface:
    def __init__(self):
        self.metric = None
        self.area = None
        self.MRT_INELIGIBLE = False
        self.IGP_EXCLUDED = False
        self.SIMULATION_OUTGOING = False
        self.init_new_computing_router()
    def init_new_computing_router(self):
        self.UNDIRECTED = True
        self.INCOMING = False
        self.OUTGOING = False
        self.INCOMING_STORED = False
        self.OUTGOING_STORED = False
        self.IN_MRT_ISLAND = False
        self.PROCESSED = False

class Bundle:
    def __init__(self):
        self.UNDIRECTED = True
        self.OUTGOING = False
        self.INCOMING = False

class Alternate:
    def __init__(self):
```

```
        self.failed_intf = None
        self.red_or_blue = None
        self.nh_list = []
        self.fec = 'NO_ALTERNATE'
        self.prot = 'NO_PROTECTION'
        self.info = 'NONE'

class Proxy_Node_Attachment_Router:
    def __init__(self):
        self.prefix = None
        self.node = None
        self.named_proxy_cost = None
        self.min_lfin = None
        self.nh_intf_list = []

class Named_Proxy_Node:
    def __init__(self):
        self.node_id = None #this is the prefix_id
        self.node_prefix_cost_list = []
        self.lfin_list = []
        self.pnar1 = None
        self.pnar2 = None
        self.pnar_X = None
        self.pnar_Y = None
        self.blue_next_hops = []
        self.red_next_hops = []
        self.primary_next_hops = []
        self.blue_next_hops_dict = {}
        self.red_next_hops_dict = {}
        self.pnh_dict = {}
        self.alt_dict = {}

def Interface_Compare(intf_a, intf_b):
    if intf_a.metric < intf_b.metric:
        return -1
    if intf_b.metric < intf_a.metric:
        return 1
    if intf_a.remote_node.node_id < intf_b.remote_node.node_id:
        return -1
    if intf_b.remote_node.node_id < intf_a.remote_node.node_id:
        return 1
    return 0

def Sort_Interfaces(topo):
    for node in topo.island_node_list:
        node.island_intf_list.sort(Interface_Compare)

def Reset_Computed_Node_and_Intf_Values(topo):
```

```
topo.init_new_computing_router()
for node in topo.node_list:
    node.init_new_computing_router()
    for intf in node.intf_list:
        intf.init_new_computing_router()

# This function takes a file with links represented by 2-digit
# numbers in the format:
# 01,05,10
# 05,02,30
# 02,01,15
# which represents a triangle topology with nodes 01, 05, and 02
# and symmetric metrics of 10, 30, and 15.

# Inclusion of a fourth column makes the metrics for the link
# asymmetric. An entry of:
# 02,07,10,15
# creates a link from node 02 to 07 with metrics 10 and 15.
def Create_Topology_From_File(filename):
    topo = Topology()
    node_id_set= set()
    cols_list = []
    # on first pass just create nodes
    with open(filename + '.csv') as topo_file:
        for line in topo_file:
            line = line.rstrip('\r\n')
            cols=line.split(',')
            cols_list.append(cols)
            nodea_node_id = int(cols[0])
            nodeb_node_id = int(cols[1])
            if (nodea_node_id > 999 or nodeb_node_id > 999):
                print("node_id must be between 0 and 999.")
                print("exiting.")
                exit()
            node_id_set.add(nodea_node_id)
            node_id_set.add(nodeb_node_id)
    for node_id in node_id_set:
        node = Node()
        node.node_id = node_id
        topo.node_list.append(node)
        topo.node_dict[node_id] = node
    # on second pass create interfaces
    for cols in cols_list:
        nodea_node_id = int(cols[0])
        nodeb_node_id = int(cols[1])
        metric = int(cols[2])
        reverse_metric = int(cols[2])
        if len(cols) > 3:
```

```
        reverse_metric=int(cols[3])
        nodea = topo.node_dict[nodea_node_id]
        nodeb = topo.node_dict[nodeb_node_id]
        nodea_intf = Interface()
        nodea_intf.metric = metric
        nodea_intf.area = 0
        nodeb_intf = Interface()
        nodeb_intf.metric = reverse_metric
        nodeb_intf.area = 0
        nodea_intf.remote_intf = nodeb_intf
        nodeb_intf.remote_intf = nodea_intf
        nodea_intf.remote_node = nodeb
        nodeb_intf.remote_node = nodea
        nodea_intf.local_node = nodea
        nodeb_intf.local_node = nodeb
        nodea_intf.link_data = len(nodea_intf_list)
        nodeb_intf.link_data = len(nodeb_intf_list)
        nodea_intf_list.append(nodea_intf)
        nodeb_intf_list.append(nodeb_intf)
    return topo

def MRT_Island_Identification(topo, computing_rtr, profile_id, area):
    if profile_id in computing_rtr.profile_id_list:
        computing_rtr.IN_MRT_ISLAND = True
        explore_list = [computing_rtr]
    else:
        return
    while explore_list != []:
        next_rtr = explore_list.pop()
        for intf in next_rtr.intf_list:
            if ( (not intf.MRT_INELIGIBLE)
                and (not intf.remote_intf.MRT_INELIGIBLE)
                and (not intf.IGP_EXCLUDED) and intf.area == area
                and (profile_id in intf.remote_node.profile_id_list)):
                intf.IN_MRT_ISLAND = True
                intf.remote_intf.IN_MRT_ISLAND = True
                if (not intf.remote_node.IN_MRT_ISLAND):
                    intf.remote_node.IN_MRT_ISLAND = True
                    explore_list.append(intf.remote_node)

def Compute_Island_Node_List_For_Test_GR(topo, test_gr):
    Reset_Computed_Node_and_Intf_Values(topo)
    topo.test_gr = topo.node_dict[test_gr]
    MRT_Island_Identification(topo, topo.test_gr, 0, 0)
    for node in topo.node_list:
        if node.IN_MRT_ISLAND:
            topo.island_node_list_for_test_gr.append(node)
```

```
def Set_Island_Intf_and_Node_Lists(topo):
    for node in topo.node_list:
        if node.IN_MRT_ISLAND:
            topo.island_node_list.append(node)
            for intf in node.intf_list:
                if intf.IN_MRT_ISLAND:
                    node.island_intf_list.append(intf)

global_dfs_number = None

def Lowpoint_Visit(x, parent, intf_p_to_x):
    global global_dfs_number
    x.dfs_number = global_dfs_number
    x.lowpoint_number = x.dfs_number
    global_dfs_number += 1
    x.dfs_parent = parent
    if intf_p_to_x == None:
        x.dfs_parent_intf = None
    else:
        x.dfs_parent_intf = intf_p_to_x.remote_intf
    x.lowpoint_parent = None
    if parent != None:
        parent.dfs_child_list.append(x)
    for intf in x.island_intf_list:
        if intf.remote_node.dfs_number == None:
            Lowpoint_Visit(intf.remote_node, x, intf)
            if intf.remote_node.lowpoint_number < x.lowpoint_number:
                x.lowpoint_number = intf.remote_node.lowpoint_number
                x.lowpoint_parent = intf.remote_node
                x.lowpoint_parent_intf = intf
        else:
            if intf.remote_node is not parent:
                if intf.remote_node.dfs_number < x.lowpoint_number:
                    x.lowpoint_number = intf.remote_node.dfs_number
                    x.lowpoint_parent = intf.remote_node
                    x.lowpoint_parent_intf = intf

def Run_Lowpoint(topo):
    global global_dfs_number
    global_dfs_number = 0
    Lowpoint_Visit(topo.gadag_root, None, None)

max_block_id = None

def Assign_Block_ID(x, cur_block_id):
    global max_block_id
    x.block_id = cur_block_id
    for c in x.dfs_child_list:
```

```
    if (c.localroot is x):
        max_block_id += 1
        Assign_Block_ID(c, max_block_id)
    else:
        Assign_Block_ID(c, cur_block_id)

def Run_Assign_Block_ID(topo):
    global max_block_id
    max_block_id = 0
    Assign_Block_ID(topo.gadag_root, max_block_id)

def Construct_Ear(x, stack, intf, ear_type):
    ear_list = []
    cur_intf = intf
    not_done = True
    while not_done:
        cur_intf.UNDIRECTED = False
        cur_intf.OUTGOING = True
        cur_intf.remote_intf.UNDIRECTED = False
        cur_intf.remote_intf.INCOMING = True
        if cur_intf.remote_node.IN_GADAG == False:
            cur_intf.remote_node.IN_GADAG = True
            ear_list.append(cur_intf.remote_node)
            if ear_type == 'CHILD':
                cur_intf = cur_intf.remote_node.lowpoint_parent_intf
            else:
                assert ear_type == 'NEIGHBOR'
                cur_intf = cur_intf.remote_node.dfs_parent_intf
        else:
            not_done = False

    if ear_type == 'CHILD' and cur_intf.remote_node is x:
        # x is a cut-vertex and the local root for the block
        # in which the ear is computed
        x.IS_CUT_VERTEX = True
        localroot = x
    else:
        # inherit local root from the end of the ear
        localroot = cur_intf.remote_node.localroot

    while ear_list != []:
        y = ear_list.pop()
        y.localroot = localroot
        stack.append(y)

def Construct_GADAG_via_Lowpoint(topo):
    gadag_root = topo.gadag_root
    gadag_root.IN_GADAG = True
```

```

gadag_root.localroot = None
stack = []
stack.append(gadag_root)
while stack != []:
    x = stack.pop()
    for intf in x.island_intf_list:
        if ( intf.remote_node.IN_GADAG == False
            and intf.remote_node.dfs_parent is x ):
            Construct_Ear(x, stack, intf, 'CHILD' )
    for intf in x.island_intf_list:
        if (intf.remote_node.IN_GADAG == False
            and intf.remote_node.dfs_parent is not x):
            Construct_Ear(x, stack, intf, 'NEIGHBOR')

def Assign_Remaining_Lowpoint_Parents(topo):
    for node in topo.island_node_list:
        if ( node is not topo.gadag_root
            and node.lowpoint_parent == None ):
            node.lowpoint_parent = node.dfs_parent
            node.lowpoint_parent_intf = node.dfs_parent_intf
            node.lowpoint_number = node.dfs_parent.dfs_number

def Add_Undirected_Block_Root_Links(topo):
    for node in topo.island_node_list:
        if node.IS_CUT_VERTEX or node is topo.gadag_root:
            for intf in node.island_intf_list:
                if ( intf.remote_node.localroot is not node
                    or intf.PROCESSED ):
                    continue
            bundle_list = []
            bundle = Bundle()
            for intf2 in node.island_intf_list:
                if intf2.remote_node is intf.remote_node:
                    bundle_list.append(intf2)
                    if not intf2.UNDIRECTED:
                        bundle.UNDIRECTED = False
                        if intf2.INCOMING:
                            bundle.INCOMING = True
                        if intf2.OUTGOING:
                            bundle.OUTGOING = True
            if bundle.UNDIRECTED:
                for intf3 in bundle_list:
                    intf3.UNDIRECTED = False
                    intf3.remote_intf.UNDIRECTED = False
                    intf3.PROCESSED = True
                    intf3.remote_intf.PROCESSED = True
                    intf3.OUTGOING = True
                    intf3.remote_intf.INCOMING = True

```

```
else:
    if (bundle.OUTGOING and bundle.INCOMING):
        for intf3 in bundle_list:
            intf3.UNDIRECTED = False
            intf3.remote_intf.UNDIRECTED = False
            intf3.PROCESSED = True
            intf3.remote_intf.PROCESSED = True
            intf3.OUTGOING = True
            intf3.INCOMING = True
            intf3.remote_intf.INCOMING = True
            intf3.remote_intf.OUTGOING = True
    elif bundle.OUTGOING:
        for intf3 in bundle_list:
            intf3.UNDIRECTED = False
            intf3.remote_intf.UNDIRECTED = False
            intf3.PROCESSED = True
            intf3.remote_intf.PROCESSED = True
            intf3.OUTGOING = True
            intf3.remote_intf.INCOMING = True
    elif bundle.INCOMING:
        for intf3 in bundle_list:
            intf3.UNDIRECTED = False
            intf3.remote_intf.UNDIRECTED = False
            intf3.PROCESSED = True
            intf3.remote_intf.PROCESSED = True
            intf3.INCOMING = True
            intf3.remote_intf.OUTGOING = True

def Modify_Block_Root_Incoming_Links(topo):
    for node in topo.island_node_list:
        if ( node.IS_CUT_VERTEX == True or node is topo.gadag_root ):
            for intf in node.island_intf_list:
                if intf.remote_node.localroot is node:
                    if intf.INCOMING:
                        intf.INCOMING = False
                        intf.INCOMING_STORED = True
                        intf.remote_intf.OUTGOING = False
                        intf.remote_intf.OUTGOING_STORED = True

def Revert_Block_Root_Incoming_Links(topo):
    for node in topo.island_node_list:
        if ( node.IS_CUT_VERTEX == True or node is topo.gadag_root ):
            for intf in node.island_intf_list:
                if intf.remote_node.localroot is node:
                    if intf.INCOMING_STORED:
                        intf.INCOMING = True
                        intf.remote_intf.OUTGOING = True
                        intf.INCOMING_STORED = False
```

```
        intf.remote_intf.OUTGOING_STORED = False

def Run_Topological_Sort_GADAG(topo):
    Modify_Block_Root_Incoming_Links(topo)
    for node in topo.island_node_list:
        node.unvisited = 0
        for intf in node.island_intf_list:
            if (intf.INCOMING == True):
                node.unvisited += 1
    working_list = []
    topo_order_list = []
    working_list.append(topo.gadag_root)
    while working_list != []:
        y = working_list.pop(0)
        topo_order_list.append(y)
        for intf in y.island_intf_list:
            if ( intf.OUTGOING == True):
                intf.remote_node.unvisited -= 1
                if intf.remote_node.unvisited == 0:
                    working_list.append(intf.remote_node)
    next_topo_order = 1
    while topo_order_list != []:
        y = topo_order_list.pop(0)
        y.topo_order = next_topo_order
        next_topo_order += 1
    Revert_Block_Root_Incoming_Links(topo)

def Set_Other_Undirected_Links_Based_On_Topo_Order(topo):
    for node in topo.island_node_list:
        for intf in node.island_intf_list:
            if intf.UNDIRECTED:
                if node.topo_order < intf.remote_node.topo_order:
                    intf.OUTGOING = True
                    intf.UNDIRECTED = False
                    intf.remote_intf.INCOMING = True
                    intf.remote_intf.UNDIRECTED = False
                else:
                    intf.INCOMING = True
                    intf.UNDIRECTED = False
                    intf.remote_intf.OUTGOING = True
                    intf.remote_intf.UNDIRECTED = False

def Initialize_Temporary_Interface_Flags(topo):
    for node in topo.island_node_list:
        for intf in node.island_intf_list:
            intf.PROCESSED = False
            intf.INCOMING_STORED = False
            intf.OUTGOING_STORED = False
```

```
def Add_Undirected_Links(topo):
    Initialize_Temporary_Interface_Flags(topo)
    Add_Undirected_Block_Root_Links(topo)
    Run_Topological_Sort_GADAG(topo)
    Set_Other_Undirected_Links_Based_On_Topo_Order(topo)

def In_Common_Block(x,y):
    if ( (x.block_id == y.block_id)
        or ( x is y.localroot) or (y is x.localroot) ):
        return True
    return False

def Copy_List_Items(target_list, source_list):
    del target_list[:] # Python idiom to remove all elements of a list
    for element in source_list:
        target_list.append(element)

def Add_Item_To_List_If_New(target_list, item):
    if item not in target_list:
        target_list.append(item)

def Store_Results(y, direction):
    if direction == 'INCREASING':
        y.HIGHER = True
        Copy_List_Items(y.blue_next_hops, y.next_hops)
    if direction == 'DECREASING':
        y.LOWER = True
        Copy_List_Items(y.red_next_hops, y.next_hops)
    if direction == 'NORMAL_SPF':
        y.primary_spf_metric = y.spf_metric
        Copy_List_Items(y.primary_next_hops, y.next_hops)
    if direction == 'MRT_ISLAND_SPF':
        Copy_List_Items(y.mrt_island_next_hops, y.next_hops)
    if direction == 'COLLAPSED_SPF':
        y.collapsed_metric = y.spf_metric
        Copy_List_Items(y.collapsed_next_hops, y.next_hops)

# Note that the Python heapq function allows for duplicate items,
# so we use the 'spf_visited' property to only consider a node
# as min_node the first time it gets removed from the heap.
def SPF_No_Traverse_Block_Root(topo, spf_root, block_root, direction):
    spf_heap = []
    for y in topo.island_node_list:
        y.spf_metric = 2147483647 # 2^31-1
        y.next_hops = []
        y.spf_visited = False
    spf_root.spf_metric = 0
    heapq.heappush(spf_heap,
```

```

        (spf_root.spf_metric, spf_root.node_id, spf_root) )
while spf_heap != []:
    #extract third element of tuple popped from heap
    min_node = heapq.heappop(spf_heap)[2]
    if min_node.spf_visited:
        continue
    min_node.spf_visited = True
    Store_Results(min_node, direction)
    if ( (min_node is spf_root) or (min_node is not block_root) ):
        for intf in min_node.island_intf_list:
            if ( ( direction == 'INCREASING' and intf.OUTGOING )
                or (direction == 'DECREASING' and intf.INCOMING ) )
                and In_Common_Block(spf_root, intf.remote_node) ):
                path_metric = min_node.spf_metric + intf.metric
                if path_metric < intf.remote_node.spf_metric:
                    intf.remote_node.spf_metric = path_metric
                    if min_node is spf_root:
                        intf.remote_node.next_hops = [intf]
                    else:
                        Copy_List_Items(intf.remote_node.next_hops,
                                       min_node.next_hops)
                heapq.heappush(spf_heap,
                               ( intf.remote_node.spf_metric,
                                 intf.remote_node.node_id,
                                 intf.remote_node ) )
            elif path_metric == intf.remote_node.spf_metric:
                if min_node is spf_root:
                    Add_Item_To_List_If_New(
                        intf.remote_node.next_hops,intf)
                else:
                    for nh_intf in min_node.next_hops:
                        Add_Item_To_List_If_New(
                            intf.remote_node.next_hops,nh_intf)

def Normal_SPF(topo, spf_root):
    spf_heap = []
    for y in topo.node_list:
        y.spf_metric = 2147483647 # 2^31-1 as max metric
        y.next_hops = []
        y.primary_spf_metric = 2147483647
        y.primary_next_hops = []
        y.spf_visited = False
    spf_root.spf_metric = 0
    heapq.heappush(spf_heap,
                   (spf_root.spf_metric,spf_root.node_id,spf_root) )
    while spf_heap != []:
        #extract third element of tuple popped from heap
        min_node = heapq.heappop(spf_heap)[2]

```

```

    if min_node.spf_visited:
        continue
    min_node.spf_visited = True
    Store_Results(min_node, 'NORMAL_SPF')
    for intf in min_node.intf_list:
        path_metric = min_node.spf_metric + intf.metric
        if path_metric < intf.remote_node.spf_metric:
            intf.remote_node.spf_metric = path_metric
            if min_node is spf_root:
                intf.remote_node.next_hops = [intf]
            else:
                Copy_List_Items(intf.remote_node.next_hops,
                                min_node.next_hops)
        heapq.heappush(spf_heap,
                      ( intf.remote_node.spf_metric,
                        intf.remote_node.node_id,
                        intf.remote_node ) )
        elif path_metric == intf.remote_node.spf_metric:
            if min_node is spf_root:
                Add_Item_To_List_If_New(
                    intf.remote_node.next_hops,intf)
            else:
                for nh_intf in min_node.next_hops:
                    Add_Item_To_List_If_New(
                        intf.remote_node.next_hops,nh_intf)

def Set_Edge(y):
    if (y.blue_next_hops == [] and y.red_next_hops == []):
        Set_Edge(y.localroot)
        Copy_List_Items(y.blue_next_hops,y.localroot.blue_next_hops)
        Copy_List_Items(y.red_next_hops ,y.localroot.red_next_hops)
        y.order_proxy = y.localroot.order_proxy

def Compute_MRT_NH_For_One_Src_To_Island_Dests(topo,x):
    for y in topo.island_node_list:
        y.HIGHER = False
        y.LOWER = False
        y.red_next_hops = []
        y.blue_next_hops = []
        y.order_proxy = y
    SPF_No_Traverse_Block_Root(topo, x, x.localroot, 'INCREASING')
    SPF_No_Traverse_Block_Root(topo, x, x.localroot, 'DECREASING')
    for y in topo.island_node_list:
        if ( y is not x and (y.block_id == x.block_id) ):
            assert (not ( y is x.localroot or x is y.localroot) )
            assert(not (y.HIGHER and y.LOWER) )
            if y.HIGHER == True:
                Copy_List_Items(y.red_next_hops,

```

```

        x.localroot.red_next_hops)
    elif y.LOWER == True:
        Copy_List_Items(y.blue_next_hops,
            x.localroot.blue_next_hops)
    else:
        Copy_List_Items(y.blue_next_hops,
            x.localroot.red_next_hops)
        Copy_List_Items(y.red_next_hops,
            x.localroot.blue_next_hops)

# Inherit x's MRT next-hops to reach the GADAG root
# from x's MRT next-hops to reach its local root,
# but first check if x is the gadag_root (in which case
# x does not have a local root) or if x's local root
# is the gadag root (in which case we already have the
# x's MRT next-hops to reach the gadag root)
if x is not topo.gadag_root and x.localroot is not topo.gadag_root:
    Copy_List_Items(topo.gadag_root.blue_next_hops,
        x.localroot.blue_next_hops)
    Copy_List_Items(topo.gadag_root.red_next_hops,
        x.localroot.red_next_hops)
    topo.gadag_root.order_proxy = x.localroot

# Inherit next-hops and order_proxies to other blocks
for y in topo.island_node_list:
    if (y is not topo.gadag_root and y is not x ):
        Set_Edge(y)

def Store_MRT_Nexthops_For_One_Src_To_Island_Dests(topo,x):
    for y in topo.island_node_list:
        if y is x:
            continue
        x.blue_next_hops_dict[y.node_id] = []
        x.red_next_hops_dict[y.node_id] = []
        Copy_List_Items(x.blue_next_hops_dict[y.node_id],
            y.blue_next_hops)
        Copy_List_Items(x.red_next_hops_dict[y.node_id],
            y.red_next_hops)

def Store_Primary_and_Alts_For_One_Src_To_Island_Dests(topo,x):
    for y in topo.island_node_list:
        x.pnh_dict[y.node_id] = []
        Copy_List_Items(x.pnh_dict[y.node_id], y.primary_next_hops)
        x.alt_dict[y.node_id] = []
        Copy_List_Items(x.alt_dict[y.node_id], y.alt_list)

def Store_Primary_NHs_For_One_Source_To_Nodes(topo,x):
    for y in topo.node_list:

```

```
x.pnh_dict[y.node_id] = []
Copy_List_Items(x.pnh_dict[y.node_id], y.primary_next_hops)

def Store_MRT_NHs_For_One_Src_To_Named_Proxy_Nodes(topo,x):
    for prefix in topo.named_proxy_dict:
        P = topo.named_proxy_dict[prefix]
        x.blue_next_hops_dict[P.node_id] = []
        x.red_next_hops_dict[P.node_id] = []
        Copy_List_Items(x.blue_next_hops_dict[P.node_id],
            P.blue_next_hops)
        Copy_List_Items(x.red_next_hops_dict[P.node_id],
            P.red_next_hops)

def Store_Alts_For_One_Src_To_Named_Proxy_Nodes(topo,x):
    for prefix in topo.named_proxy_dict:
        P = topo.named_proxy_dict[prefix]
        x.alt_dict[P.node_id] = []
        Copy_List_Items(x.alt_dict[P.node_id],
            P.alt_list)

def Store_Primary_NHs_For_One_Src_To_Named_Proxy_Nodes(topo,x):
    for prefix in topo.named_proxy_dict:
        P = topo.named_proxy_dict[prefix]
        x.pnh_dict[P.node_id] = []
        Copy_List_Items(x.pnh_dict[P.node_id],
            P.primary_next_hops)

def Select_Alternates_Internal(D, F, primary_intf,
                               D_lower, D_higher, D_topo_order):
    if D_higher and D_lower:
        if F.HIGHER and F.LOWER:
            if F.topo_order > D_topo_order:
                return 'USE_BLUE'
            else:
                return 'USE_RED'
        if F.HIGHER:
            return 'USE_RED'
        if F.LOWER:
            return 'USE_BLUE'
        assert(primary_intf.MRT_INELIGIBLE
            or primary_intf.remote_intf.MRT_INELIGIBLE)
        return 'USE_RED_OR_BLUE'
    if D_higher:
        if F.HIGHER and F.LOWER:
            return 'USE_BLUE'
        if F.LOWER:
            return 'USE_BLUE'
        if F.HIGHER:
```

```
        if (F.topo_order > D_topo_order):
            return 'USE_BLUE'
        if (F.topo_order < D_topo_order):
            return 'USE_RED'
        assert(False)
    assert(primary_intf.MRT_INELIGIBLE
           or primary_intf.remote_intf.MRT_INELIGIBLE)
    return 'USE_RED_OR_BLUE'
if D_lower:
    if F.HIGHER and F.LOWER:
        return 'USE_RED'
    if F.HIGHER:
        return 'USE_RED'
    if F.LOWER:
        if F.topo_order > D_topo_order:
            return 'USE_BLUE'
        if F.topo_order < D_topo_order:
            return 'USE_RED'
        assert(False)
    assert(primary_intf.MRT_INELIGIBLE
           or primary_intf.remote_intf.MRT_INELIGIBLE)
    return 'USE_RED_OR_BLUE'
else: # D is unordered wrt S
    if F.HIGHER and F.LOWER:
        if primary_intf.OUTGOING and primary_intf.INCOMING:
            # This can happen when F and D are in different blocks
            return 'USE_RED_OR_BLUE'
        if primary_intf.OUTGOING:
            return 'USE_BLUE'
        if primary_intf.INCOMING:
            return 'USE_RED'
        #This can occur when primary_intf is MRT_INELIGIBLE.
        #This appears to be a case where the special
        #construction of the GADAG allows us to choose red,
        #whereas with an arbitrary GADAG, neither red nor blue
        #is guaranteed to work.
        assert(primary_intf.MRT_INELIGIBLE
               or primary_intf.remote_intf.MRT_INELIGIBLE)
        return 'USE_RED'
    if F.LOWER:
        return 'USE_RED'
    if F.HIGHER:
        return 'USE_BLUE'
    assert(primary_intf.MRT_INELIGIBLE
           or primary_intf.remote_intf.MRT_INELIGIBLE)
    if F.topo_order > D_topo_order:
        return 'USE_BLUE'
    else:
```

```
        return 'USE_RED'

def Select_Alternates(D, F, primary_intf):
    S = primary_intf.local_node
    if not In_Common_Block(F, S):
        return 'PRIM_NH_IN_DIFFERENT_BLOCK'
    if (D is F) or (D.order_proxy is F):
        return 'PRIM_NH_IS_D_OR_OP_FOR_D'
    D_lower = D.order_proxy.LOWER
    D_higher = D.order_proxy.HIGHER
    D_topo_order = D.order_proxy.topo_order
    return Select_Alternates_Internal(D, F, primary_intf,
                                     D_lower, D_higher, D_topo_order)

def Is_Remote_Node_In_NH_List(node, intf_list):
    for intf in intf_list:
        if node is intf.remote_node:
            return True
    return False

def Select_Alts_For_One_Src_To_Island_Dests(topo,x):
    Normal_SPF(topo, x)
    for D in topo.island_node_list:
        D.alt_list = []
        if D is x:
            continue
        for failed_intf in D.primary_next_hops:
            alt = Alternate()
            alt.failed_intf = failed_intf
            cand_alt_list = []
            F = failed_intf.remote_node
            #We need to test if F is in the island, as opposed
            #to just testing if failed_intf is in island_intf_list,
            #because failed_intf could be marked as MRT_INELIGIBLE.
            if F in topo.island_node_list:
                alt.info = Select_Alternates(D, F, failed_intf)
            else:
                #The primary next-hop is not in the MRT Island.
                #Either red or blue will avoid the primary next-hop,
                #because the primary next-hop is not even in the
                #GADAG.
                alt.info = 'USE_RED_OR_BLUE'

            if (alt.info == 'USE_RED_OR_BLUE'):
                alt.red_or_blue = random.choice(['USE_RED', 'USE_BLUE'])
            if (alt.info == 'USE_BLUE'
                or alt.red_or_blue == 'USE_BLUE'):
```

```

Copy_List_Items(alt.nh_list, D.blue_next_hops)
alt.fec = 'BLUE'
alt.prot = 'NODE_PROTECTION'
if (alt.info == 'USE_RED' or alt.red_or_blue == 'USE_RED'):
Copy_List_Items(alt.nh_list, D.red_next_hops)
alt.fec = 'RED'
alt.prot = 'NODE_PROTECTION'
if (alt.info == 'PRIM_NH_IN_DIFFERENT_BLOCK'):
alt.fec = 'NO_ALTERNATE'
alt.prot = 'NO_PROTECTION'
if (alt.info == 'PRIM_NH_IS_D_OR_OP_FOR_D'):
if failed_intf.OUTGOING and failed_intf.INCOMING:
# cut-link: if there are parallel cut links, use
# the link(s) with lowest metric that are not
# primary intf or None
cand_alt_list = [None]
min_metric = 2147483647
for intf in x.island_intf_list:
if ( intf is not failed_intf and
(intf.remote_node is
failed_intf.remote_node)):
if intf.metric < min_metric:
cand_alt_list = [intf]
min_metric = intf.metric
elif intf.metric == min_metric:
cand_alt_list.append(intf)
if cand_alt_list != [None]:
alt.fec = 'GREEN'
alt.prot = 'PARALLEL_CUTLINK'
else:
alt.fec = 'NO_ALTERNATE'
alt.prot = 'NO_PROTECTION'
Copy_List_Items(alt.nh_list, cand_alt_list)

# Is_Remote_Node_In_NH_List() is used, as opposed
# to just checking if failed_intf is in D.red_next_hops,
# because failed_intf could be marked as MRT_INELIGIBLE.
elif Is_Remote_Node_In_NH_List(F, D.red_next_hops):
Copy_List_Items(alt.nh_list, D.blue_next_hops)
alt.fec = 'BLUE'
alt.prot = 'LINK_PROTECTION'
elif Is_Remote_Node_In_NH_List(F, D.blue_next_hops):
Copy_List_Items(alt.nh_list, D.red_next_hops)
alt.fec = 'RED'
alt.prot = 'LINK_PROTECTION'
else:
alt.fec = random.choice(['RED', 'BLUE'])
alt.prot = 'LINK_PROTECTION'

```

```

        D.alt_list.append(alt)

def Write_GADAG_To_File(topo, file_prefix):
    gadag_edge_list = []
    for node in topo.node_list:
        for intf in node.intf_list:
            if intf.SIMULATION_OUTGOING:
                local_node = "%04d" % (intf.local_node.node_id)
                remote_node = "%04d" % (intf.remote_node.node_id)
                intf_data = "%03d" % (intf.link_data)
                edge_string=(local_node+', '+remote_node+', '+
                    intf_data+'\n')
                gadag_edge_list.append(edge_string)
    gadag_edge_list.sort();
    filename = file_prefix + '_gadag.csv'
    with open(filename, 'w') as gadag_file:
        gadag_file.write('local_node,'\
            'remote_node,local_intf_link_data\n')
        for edge_string in gadag_edge_list:
            gadag_file.write(edge_string);

def Write_MRTs_For_All_Dests_To_File(topo, color, file_prefix):
    edge_list = []
    for node in topo.island_node_list_for_test_gr:
        if color == 'blue':
            node_next_hops_dict = node.blue_next_hops_dict
        elif color == 'red':
            node_next_hops_dict = node.red_next_hops_dict
        for dest_node_id in node_next_hops_dict:
            for intf in node_next_hops_dict[dest_node_id]:
                gadag_root = "%04d" % (topo.gadag_root.node_id)
                dest_node = "%04d" % (dest_node_id)
                local_node = "%04d" % (intf.local_node.node_id)
                remote_node = "%04d" % (intf.remote_node.node_id)
                intf_data = "%03d" % (intf.link_data)
                edge_string=(gadag_root+', '+dest_node+', '+local_node+
                    ', '+remote_node+', '+intf_data+'\n')
                edge_list.append(edge_string)
    edge_list.sort()
    filename = file_prefix + '_' + color + '_to_all.csv'
    with open(filename, 'w') as mrt_file:
        mrt_file.write('gadag_root,dest,'\
            'local_node,remote_node,link_data\n')
        for edge_string in edge_list:
            mrt_file.write(edge_string);

def Write_Both_MRTs_For_All_Dests_To_File(topo, file_prefix):
    Write_MRTs_For_All_Dests_To_File(topo, 'blue', file_prefix)

```

```

Write_MRTs_For_All_Dests_To_File(topo, 'red', file_prefix)

def Write_Alternates_For_All_Dests_To_File(topo, file_prefix):
    edge_list = []
    for x in topo.island_node_list_for_test_gr:
        for dest_node_id in x.alt_dict:
            alt_list = x.alt_dict[dest_node_id]
            for alt in alt_list:
                for alt_intf in alt.nh_list:
                    gadag_root = "%04d" % (topo.gadag_root.node_id)
                    dest_node = "%04d" % (dest_node_id)
                    prim_local_node = \
                        "%04d" % (alt.failed_intf.local_node.node_id)
                    prim_remote_node = \
                        "%04d" % (alt.failed_intf.remote_node.node_id)
                    prim_intf_data = \
                        "%03d" % (alt.failed_intf.link_data)
                    if alt_intf == None:
                        alt_local_node = "None"
                        alt_remote_node = "None"
                        alt_intf_data = "None"
                    else:
                        alt_local_node = \
                            "%04d" % (alt_intf.local_node.node_id)
                        alt_remote_node = \
                            "%04d" % (alt_intf.remote_node.node_id)
                        alt_intf_data = \
                            "%03d" % (alt_intf.link_data)
                    edge_string = (gadag_root+', '+dest_node+', '+
                        prim_local_node+', '+prim_remote_node+', '+
                        prim_intf_data+', '+alt_local_node+', '+
                        alt_remote_node+', '+alt_intf_data+', '+
                        alt.fec + '\n')
                    edge_list.append(edge_string)
    edge_list.sort()
    filename = file_prefix + '_alts_to_all.csv'
    with open(filename, 'w') as alt_file:
        alt_file.write('gadag_root,dest,'\
            'prim_nh.local_node,prim_nh.remote_node,'\
            'prim_nh.link_data,alt_nh.local_node,'\
            'alt_nh.remote_node,alt_nh.link_data,'\
            'alt_nh.fec\n')
        for edge_string in edge_list:
            alt_file.write(edge_string);

def Raise_GADAG_Root_Selection_Priority(topo,node_id):
    node = topo.node_dict[node_id]
    node.GR_sel_priority = 255

```

```
def Lower_GADAG_Root_Selection_Priority(topo,node_id):
    node = topo.node_dict[node_id]
    node.GR_sel_priority = 128

def GADAG_Root_Compare(node_a, node_b):
    if (node_a.GR_sel_priority > node_b.GR_sel_priority):
        return 1
    elif (node_a.GR_sel_priority < node_b.GR_sel_priority):
        return -1
    else:
        if node_a.node_id > node_b.node_id:
            return 1
        elif node_a.node_id < node_b.node_id:
            return -1

def Set_GADAG_Root(topo,computing_router):
    gadag_root_list = []
    for node in topo.island_node_list:
        gadag_root_list.append(node)
    gadag_root_list.sort(GADAG_Root_Compare)
    topo.gadag_root = gadag_root_list.pop()

def Add_Prefix_Advertisements_From_File(topo, filename):
    prefix_filename = filename + '.prefix'
    cols_list = []
    if not os.path.exists(prefix_filename):
        return
    with open(prefix_filename) as prefix_file:
        for line in prefix_file:
            line = line.rstrip('\r\n')
            cols=line.split(',')
            cols_list.append(cols)
            prefix_id = int(cols[0])
            if prefix_id < 2000 or prefix_id >2999:
                print('skipping the following line of prefix file')
                print('prefix id should be between 2000 and 2999')
                print(line)
                continue
            prefix_node_id = int(cols[1])
            prefix_cost = int(cols[2])
            advertising_node = topo.node_dict[prefix_node_id]
            advertising_node.prefix_cost_dict[prefix_id] = prefix_cost

def Add_Prefixes_for_Non_Island_Nodes(topo):
    for node in topo.node_list:
        if node.IN_MRT_ISLAND:
            continue
        prefix_id = node.node_id + 1000
```

```

        node.prefix_cost_dict[prefix_id] = 0

def Add_Profile_IDs_from_File(topo, filename):
    profile_filename = filename + '.profile'
    for node in topo.node_list:
        node.profile_id_list = []
    cols_list = []
    if os.path.exists(profile_filename):
        with open(profile_filename) as profile_file:
            for line in profile_file:
                line = line.rstrip('\r\n')
                cols=line.split(',')
                cols_list.append(cols)
                node_id = int(cols[0])
                profile_id = int(cols[1])
                this_node = topo.node_dict[node_id]
                this_node.profile_id_list.append(profile_id)
    else:
        for node in topo.node_list:
            node.profile_id_list = [0]

def Island_Marking_SPF(topo,spf_root):
    spf_root.isl_marking_spf_dict = {}
    for y in topo.node_list:
        y.spf_metric = 2147483647 # 2^31-1 as max metric
        y.PATH_HITS_ISLAND = False
        y.next_hops = []
        y.spf_visited = False
    spf_root.spf_metric = 0
    spf_heap = []
    heapq.heappush(spf_heap,
                   (spf_root.spf_metric,spf_root.node_id,spf_root) )
    while spf_heap != []:
        #extract third element of tuple popped from heap
        min_node = heapq.heappop(spf_heap)[2]
        if min_node.spf_visited:
            continue
        min_node.spf_visited = True
        spf_root.isl_marking_spf_dict[min_node.node_id] = \
            (min_node.spf_metric, min_node.PATH_HITS_ISLAND)
        for intf in min_node.intf_list:
            path_metric = min_node.spf_metric + intf.metric
            if path_metric < intf.remote_node.spf_metric:
                intf.remote_node.spf_metric = path_metric
            if min_node is spf_root:
                intf.remote_node.next_hops = [intf]
            else:
                Copy_List_Items(intf.remote_node.next_hops,

```

```

        min_node.next_hops)
    if (intf.remote_node.IN_MRT_ISLAND):
        intf.remote_node.PATH_HITS_ISLAND = True
    else:
        intf.remote_node.PATH_HITS_ISLAND = \
            min_node.PATH_HITS_ISLAND
    heapq.heappush(spf_heap,
        ( intf.remote_node.spf_metric,
          intf.remote_node.node_id,
          intf.remote_node ) )
elif path_metric == intf.remote_node.spf_metric:
    if min_node is spf_root:
        Add_Item_To_List_If_New(
            intf.remote_node.next_hops,intf)
    else:
        for nh_intf in min_node.next_hops:
            Add_Item_To_List_If_New(
                intf.remote_node.next_hops,nh_intf)
    if (intf.remote_node.IN_MRT_ISLAND):
        intf.remote_node.PATH_HITS_ISLAND = True
    else:
        if (intf.remote_node.PATH_HITS_ISLAND
            or min_node.PATH_HITS_ISLAND):
            intf.remote_node.PATH_HITS_ISLAND = True

def Create_Basic_Named_Proxy_Nodes(topo):
    for node in topo.node_list:
        for prefix in node.prefix_cost_dict:
            prefix_cost = node.prefix_cost_dict[prefix]
            if prefix in topo.named_proxy_dict:
                P = topo.named_proxy_dict[prefix]
                P.node_prefix_cost_list.append((node,prefix_cost))
            else:
                P = Named_Proxy_Node()
                topo.named_proxy_dict[prefix] = P
                P.node_id = prefix
                P.node_prefix_cost_list = [(node,prefix_cost)]

def Compute_Loop_Free_Island_Neighbors_For_Each_Prefix(topo):
    topo.island_nbr_set = set()
    topo.island_border_set = set()
    for node in topo.node_list:
        if node.IN_MRT_ISLAND:
            continue
        for intf in node.intf_list:
            if intf.remote_node.IN_MRT_ISLAND:
                topo.island_nbr_set.add(node)

```

```

        topo.island_border_set.add(intf.remote_node)

for island_nbr in topo.island_nbr_set:
    Island_Marking_SPF(topo, island_nbr)

for prefix in topo.named_proxy_dict:
    P = topo.named_proxy_dict[prefix]
    P.lfin_list = []
    for island_nbr in topo.island_nbr_set:
        min_isl_nbr_to_pref_cost = 2147483647
        for (adv_node, prefix_cost) in P.node_prefix_cost_list:
            (adv_node_cost, path_hits_island) = \
                island_nbr.isl_marking_spf_dict[adv_node.node_id]
            isl_nbr_to_pref_cost = adv_node_cost + prefix_cost
            if isl_nbr_to_pref_cost < min_isl_nbr_to_pref_cost:
                min_isl_nbr_to_pref_cost = isl_nbr_to_pref_cost
                min_path_hits_island = path_hits_island
            elif isl_nbr_to_pref_cost == min_isl_nbr_to_pref_cost:
                if min_path_hits_island or path_hits_island:
                    min_path_hits_island = True
        if not min_path_hits_island:
            P.lfin_list.append( (island_nbr,
                                min_isl_nbr_to_pref_cost) )

def Compute_Island_Border_Router_LFIN_Pairs_For_Each_Prefix(topo):
    for ibr in topo.island_border_set:
        ibr.prefix_lfin_dict = {}
        ibr.min_intf_metric_dict = {}
        ibr.min_intf_list_dict = {}
        ibr.min_intf_list_dict[None] = None
        for intf in ibr_intf_list:
            if not intf.remote_node in topo.island_nbr_set:
                continue
            if not intf.remote_node in ibr.min_intf_metric_dict:
                ibr.min_intf_metric_dict[intf.remote_node] = \
                    intf.metric
                ibr.min_intf_list_dict[intf.remote_node] = [intf]
            else:
                if (intf.metric
                    < ibr.min_intf_metric_dict[intf.remote_node]):
                    ibr.min_intf_metric_dict[intf.remote_node] = \
                        intf.metric
                    ibr.min_intf_list_dict[intf.remote_node] = [intf]
                elif (intf.metric
                    < ibr.min_intf_metric_dict[intf.remote_node]):
                    ibr.min_intf_list_dict[intf.remote_node].\
                        append(intf)

```

```

for prefix in topo.named_proxy_dict:
    P = topo.named_proxy_dict[prefix]
    for ibr in topo.island_border_set:
        min_ibr_lfin_pref_cost = 2147483647
        min_lfin = None
        for (lfin, lfin_to_pref_cost) in P.lfin_list:
            if not lfin in ibr.min_intf_metric_dict:
                continue
            ibr_lfin_pref_cost = \
                ibr.min_intf_metric_dict[lfin] + lfin_to_pref_cost
            if ibr_lfin_pref_cost < min_ibr_lfin_pref_cost:
                min_ibr_lfin_pref_cost = ibr_lfin_pref_cost
                min_lfin = lfin
        ibr.prefix_lfin_dict[prefix] = (min_lfin,
            min_ibr_lfin_pref_cost,
            ibr.min_intf_list_dict[min_lfin])

def Proxy_Node_Att_Router_Compare(pnar_a, pnar_b):
    if pnar_a.named_proxy_cost < pnar_b.named_proxy_cost:
        return -1
    if pnar_b.named_proxy_cost < pnar_a.named_proxy_cost:
        return 1
    if pnar_a.node.node_id < pnar_b.node.node_id:
        return -1
    if pnar_b.node.node_id < pnar_a.node.node_id:
        return 1
    if pnar_a.min_lfin == None:
        return -1
    if pnar_b.min_lfin == None:
        return 1

def Choose_Proxy_Node_Attachment_Routers(topo):
    for prefix in topo.named_proxy_dict:
        P = topo.named_proxy_dict[prefix]
        pnar_candidate_list = []
        for (node, prefix_cost) in P.node_prefix_cost_list:
            if not node.IN_MRT_ISLAND:
                continue
            pnar = Proxy_Node_Attachment_Router()
            pnar.prefix = prefix
            pnar.named_proxy_cost = prefix_cost
            pnar.node = node
            pnar_candidate_list.append(pnar)
        for ibr in topo.island_border_set:
            (min_lfin, prefix_cost, min_intf_list) = \
                ibr.prefix_lfin_dict[prefix]
            if min_lfin == None:
                continue

```

```

        pnar = Proxy_Node_Attachment_Router()
        pnar.named_proxy_cost = prefix_cost
        pnar.node = ibr
        pnar.min_lfin = min_lfin
        pnar.nh_intf_list = min_intf_list
        pnar_candidate_list.append(pnar)
    pnar_candidate_list.sort(cmp=Proxy_Node_Att_Router_Compare)
    #pop first element from list
    first_pnar = pnar_candidate_list.pop(0)
    second_pnar = None
    for next_pnar in pnar_candidate_list:
        if next_pnar.node is first_pnar.node:
            continue
        second_pnar = next_pnar
        break

    P.pnar1 = first_pnar
    P.pnar2 = second_pnar

def Attach_Named_Proxy_Nodes(topo):
    Compute_Loop_Free_Island_Neighbors_For_Each_Prefix(topo)
    Compute_Island_Border_Router_LFIN_Pairs_For_Each_Prefix(topo)
    Choose_Proxy_Node_Attachment_Routers(topo)

def Select_Proxy_Node_NHs(P,S):
    if P.pnar1.node.node_id < P.pnar2.node.node_id:
        X = P.pnar1.node
        Y = P.pnar2.node
    else:
        X = P.pnar2.node
        Y = P.pnar1.node
    P.pnar_X = X
    P.pnar_Y = Y
    A = X.order_proxy
    B = Y.order_proxy
    if (A is S.localroot
        and B is S.localroot):
        #print("1.0")
        Copy_List_Items(P.blue_next_hops, X.blue_next_hops)
        Copy_List_Items(P.red_next_hops, Y.red_next_hops)
        return
    if (A is S.localroot
        and B is not S.localroot):
        #print("2.0")
        if B.LOWER:
            #print("2.1")
            Copy_List_Items(P.blue_next_hops, X.blue_next_hops)
            Copy_List_Items(P.red_next_hops, Y.red_next_hops)

```

```
        return
    if B.HIGHER:
        #print("2.2")
        Copy_List_Items(P.blue_next_hops, X.red_next_hops)
        Copy_List_Items(P.red_next_hops, Y.blue_next_hops)
        return
    else:
        #print("2.3")
        Copy_List_Items(P.blue_next_hops, X.red_next_hops)
        Copy_List_Items(P.red_next_hops, Y.red_next_hops)
        return
if (A is not S.localroot
and B is S.localroot):
    #print("3.0")
    if A.LOWER:
        #print("3.1")
        Copy_List_Items(P.blue_next_hops, X.red_next_hops)
        Copy_List_Items(P.red_next_hops, Y.blue_next_hops)
        return
    if A.HIGHER:
        #print("3.2")
        Copy_List_Items(P.blue_next_hops, X.blue_next_hops)
        Copy_List_Items(P.red_next_hops, Y.red_next_hops)
        return
    else:
        #print("3.3")
        Copy_List_Items(P.blue_next_hops, X.red_next_hops)
        Copy_List_Items(P.red_next_hops, Y.red_next_hops)
        return
if (A is not S.localroot
and B is not S.localroot):
    #print("4.0")
    if (S is A.localroot or S is B.localroot):
        #print("4.05")
        if A.topo_order < B.topo_order:
            #print("4.05.1")
            Copy_List_Items(P.blue_next_hops, X.blue_next_hops)
            Copy_List_Items(P.red_next_hops, Y.red_next_hops)
            return
        else:
            #print("4.05.2")
            Copy_List_Items(P.blue_next_hops, X.red_next_hops)
            Copy_List_Items(P.red_next_hops, Y.blue_next_hops)
            return
    if A.LOWER:
        #print("4.1")
        if B.HIGHER:
            #print("4.1.1")
```

```
Copy_List_Items(P.blue_next_hops, X.red_next_hops)
Copy_List_Items(P.red_next_hops, Y.blue_next_hops)
return
if B.LOWER:
    #print("4.1.2")
    if A.topo_order < B.topo_order:
        #print("4.1.2.1")
        Copy_List_Items(P.blue_next_hops, X.blue_next_hops)
        Copy_List_Items(P.red_next_hops, Y.red_next_hops)
        return
    else:
        #print("4.1.2.2")
        Copy_List_Items(P.blue_next_hops, X.red_next_hops)
        Copy_List_Items(P.red_next_hops, Y.blue_next_hops)
        return
else:
    #print("4.1.3")
    Copy_List_Items(P.blue_next_hops, X.red_next_hops)
    Copy_List_Items(P.red_next_hops, Y.red_next_hops)
    return
if A.HIGHER:
    #print("4.2")
    if B.HIGHER:
        #print("4.2.1")
        if A.topo_order < B.topo_order:
            #print("4.2.1.1")
            Copy_List_Items(P.blue_next_hops, X.blue_next_hops)
            Copy_List_Items(P.red_next_hops, Y.red_next_hops)
            return
        else:
            #print("4.2.1.2")
            Copy_List_Items(P.blue_next_hops, X.red_next_hops)
            Copy_List_Items(P.red_next_hops, Y.blue_next_hops)
            return
    if B.LOWER:
        #print("4.2.2")
        Copy_List_Items(P.blue_next_hops, X.blue_next_hops)
        Copy_List_Items(P.red_next_hops, Y.red_next_hops)
        return
    else:
        #print("4.2.3")
        Copy_List_Items(P.blue_next_hops, X.blue_next_hops)
        Copy_List_Items(P.red_next_hops, Y.blue_next_hops)
        return
else:
    #print("4.3")
    if B.LOWER:
        #print("4.3.1")
```

```

        Copy_List_Items(P.blue_next_hops, X.red_next_hops)
        Copy_List_Items(P.red_next_hops, Y.red_next_hops)
        return
    if B.HIGHER:
        #print("4.3.2")
        Copy_List_Items(P.blue_next_hops, X.blue_next_hops)
        Copy_List_Items(P.red_next_hops, Y.blue_next_hops)
        return
    else:
        #print("4.3.3")
        if A.topo_order < B.topo_order:
            #print("4.3.3.1")
            Copy_List_Items(P.blue_next_hops, X.blue_next_hops)
            Copy_List_Items(P.red_next_hops, Y.red_next_hops)
            return
        else:
            #print("4.3.3.2")
            Copy_List_Items(P.blue_next_hops, X.red_next_hops)
            Copy_List_Items(P.red_next_hops, Y.blue_next_hops)
            return
    assert(False)

def Compute_MRT_NHs_For_One_Src_To_Named_Proxy_Nodes(topo,S):
    for prefix in topo.named_proxy_dict:
        P = topo.named_proxy_dict[prefix]
        if P.pnar2 == None:
            if S is P.pnar1.node:
                # set the MRT next-hops for the PNAR to
                # reach the LFIN and change FEC to green
                Copy_List_Items(P.blue_next_hops,
                               P.pnar1.nh_intf_list)
                S.blue_to_green_nh_dict[P.node_id] = True
                Copy_List_Items(P.red_next_hops,
                               P.pnar1.nh_intf_list)
                S.red_to_green_nh_dict[P.node_id] = True
            else:
                # inherit MRT NHs for P from pnar1
                Copy_List_Items(P.blue_next_hops,
                               P.pnar1.node.blue_next_hops)
                Copy_List_Items(P.red_next_hops,
                               P.pnar1.node.red_next_hops)
        else:
            Select_Proxy_Node_NHs(P,S)
            # set the MRT next-hops for the PNAR to reach the LFIN
            # and change FEC to green rely on the red or blue
            # next-hops being empty to figure out which one needs
            # to point to the LFIN.
            if S is P.pnar1.node:

```

```

        this_pnar = P.pnar1
    elif S is P.pnar2.node:
        this_pnar = P.pnar2
    else:
        continue
    if P.blue_next_hops == []:
        Copy_List_Items(P.blue_next_hops,
            this_pnar.nh_intf_list)
        S.blue_to_green_nh_dict[P.node_id] = True
    if P.red_next_hops == []:
        Copy_List_Items(P.red_next_hops,
            this_pnar.nh_intf_list)
        S.red_to_green_nh_dict[P.node_id] = True

def Select_Alternates_Proxy_Node(P,F,primary_intf):
    S = primary_intf.local_node
    X = P.pnar_X
    Y = P.pnar_Y
    A = X.order_proxy
    B = Y.order_proxy
    if F is A and F is B:
        return 'PRIM_NH_IS_OP_FOR_BOTH_X_AND_Y'
    if F is A:
        return 'USE_RED'
    if F is B:
        return 'USE_BLUE'

    if not In_Common_Block(A, B):
        if In_Common_Block(F, A):
            return 'USE_RED'
        elif In_Common_Block(F, B):
            return 'USE_BLUE'
        else:
            return 'USE_RED_OR_BLUE'
    if (not In_Common_Block(F, A)
        and not In_Common_Block(F, B) ):
        return 'USE_RED_OR_BLUE'

    alt_to_X = Select_Alternates(X, F, primary_intf)
    alt_to_Y = Select_Alternates(Y, F, primary_intf)

    if (alt_to_X == 'USE_RED_OR_BLUE'
        and alt_to_Y == 'USE_RED_OR_BLUE'):
        return 'USE_RED_OR_BLUE'
    if alt_to_X == 'USE_RED_OR_BLUE':
        return 'USE_BLUE'
    if alt_to_Y == 'USE_RED_OR_BLUE':
        return 'USE_RED'

```

```
if (A is S.localroot
    and B is S.localroot):
    #print("1.0")
    if (alt_to_X == 'USE_BLUE' and alt_to_Y == 'USE_RED'):
        return 'USE_RED_OR_BLUE'
    if alt_to_X == 'USE_BLUE':
        return 'USE_BLUE'
    if alt_to_Y == 'USE_RED':
        return 'USE_RED'
    assert(False)
if (A is S.localroot
    and B is not S.localroot):
    #print("2.0")
    if B.LOWER:
        #print("2.1")
        if (alt_to_X == 'USE_BLUE' and alt_to_Y == 'USE_RED'):
            return 'USE_RED_OR_BLUE'
        if alt_to_X == 'USE_BLUE':
            return 'USE_BLUE'
        if alt_to_Y == 'USE_RED':
            return 'USE_RED'
        assert(False)
    if B.HIGHER:
        #print("2.2")
        if (alt_to_X == 'USE_RED' and alt_to_Y == 'USE_BLUE'):
            return 'USE_RED_OR_BLUE'
        if alt_to_X == 'USE_RED':
            return 'USE_BLUE'
        if alt_to_Y == 'USE_BLUE':
            return 'USE_RED'
        assert(False)
    else:
        #print("2.3")
        if (alt_to_X == 'USE_RED' and alt_to_Y == 'USE_RED'):
            return 'USE_RED_OR_BLUE'
        if alt_to_X == 'USE_RED':
            return 'USE_BLUE'
        if alt_to_Y == 'USE_RED':
            return 'USE_RED'
        assert(False)
if (A is not S.localroot
    and B is S.localroot):
    #print("3.0")
    if A.LOWER:
        #print("3.1")
        if (alt_to_X == 'USE_RED' and alt_to_Y == 'USE_BLUE'):
            return 'USE_RED_OR_BLUE'
        if alt_to_X == 'USE_RED':
```

```
        return 'USE_BLUE'
    if alt_to_Y == 'USE_BLUE':
        return 'USE_RED'
    assert(False)
if A.HIGHER:
    #print("3.2")
    if (alt_to_X == 'USE_BLUE' and alt_to_Y == 'USE_RED'):
        return 'USE_RED_OR_BLUE'
    if alt_to_X == 'USE_BLUE':
        return 'USE_BLUE'
    if alt_to_Y == 'USE_RED':
        return 'USE_RED'
    assert(False)
else:
    #print("3.3")
    if (alt_to_X == 'USE_RED' and alt_to_Y == 'USE_RED'):
        return 'USE_RED_OR_BLUE'
    if alt_to_X == 'USE_RED':
        return 'USE_BLUE'
    if alt_to_Y == 'USE_RED':
        return 'USE_RED'
    assert(False)
if (A is not S.localroot
and B is not S.localroot):
    #print("4.0")
    if (S is A.localroot or S is B.localroot):
        #print("4.05")
        if A.topo_order < B.topo_order:
            #print("4.05.1")
            if (alt_to_X == 'USE_BLUE' and alt_to_Y == 'USE_RED'):
                return 'USE_RED_OR_BLUE'
            if alt_to_X == 'USE_BLUE':
                return 'USE_BLUE'
            if alt_to_Y == 'USE_RED':
                return 'USE_RED'
            assert(False)
        else:
            #print("4.05.2")
            if (alt_to_X == 'USE_RED' and alt_to_Y == 'USE_BLUE'):
                return 'USE_RED_OR_BLUE'
            if alt_to_X == 'USE_RED':
                return 'USE_BLUE'
            if alt_to_Y == 'USE_BLUE':
                return 'USE_RED'
            assert(False)
    if A.LOWER:
        #print("4.1")
        if B.HIGHER:
```

```
#print("4.1.1")
if (alt_to_X == 'USE_RED' and alt_to_Y == 'USE_BLUE'):
    return 'USE_RED_OR_BLUE'
if alt_to_X == 'USE_RED':
    return 'USE_BLUE'
if alt_to_Y == 'USE_BLUE':
    return 'USE_RED'
assert(False)
if B.LOWER:
    #print("4.1.2")
    if A.topo_order < B.topo_order:
        #print("4.1.2.1")
        if (alt_to_X == 'USE_BLUE'
            and alt_to_Y == 'USE_RED'):
            return 'USE_RED_OR_BLUE'
        if alt_to_X == 'USE_BLUE':
            return 'USE_BLUE'
        if alt_to_Y == 'USE_RED':
            return 'USE_RED'
        assert(False)
    else:
        #print("4.1.2.2")
        if (alt_to_X == 'USE_RED'
            and alt_to_Y == 'USE_BLUE'):
            return 'USE_RED_OR_BLUE'
        if alt_to_X == 'USE_RED':
            return 'USE_BLUE'
        if alt_to_Y == 'USE_BLUE':
            return 'USE_RED'
        assert(False)
else:
    #print("4.1.3")
    if (F.LOWER and not F.HIGHER
        and F.topo_order > A.topo_order):
        #print("4.1.3.1")
        return 'USE_RED'
    else:
        #print("4.1.3.2")
        return 'USE_BLUE'
if A.HIGHER:
    #print("4.2")
    if B.HIGHER:
        #print("4.2.1")
        if A.topo_order < B.topo_order:
            #print("4.2.1.1")
            if (alt_to_X == 'USE_BLUE'
                and alt_to_Y == 'USE_RED'):
                return 'USE_RED_OR_BLUE'
```

```
        if alt_to_X == 'USE_BLUE':
            return 'USE_BLUE'
        if alt_to_Y == 'USE_RED':
            return 'USE_RED'
        assert(False)
    else:
        #print("4.2.1.2")
        if (alt_to_X == 'USE_RED'
            and alt_to_Y == 'USE_BLUE'):
            return 'USE_RED_OR_BLUE'
        if alt_to_X == 'USE_RED':
            return 'USE_BLUE'
        if alt_to_Y == 'USE_BLUE':
            return 'USE_RED'
        assert(False)
if B.LOWER:
    #print("4.2.2")
    if (alt_to_X == 'USE_BLUE'
        and alt_to_Y == 'USE_RED'):
        return 'USE_RED_OR_BLUE'
    if alt_to_X == 'USE_BLUE':
        return 'USE_BLUE'
    if alt_to_Y == 'USE_RED':
        return 'USE_RED'
    assert(False)
else:
    #print("4.2.3")
    if (F.HIGHER and not F.LOWER
        and F.topo_order < A.topo_order):
        return 'USE_RED'
    else:
        return 'USE_BLUE'
else:
    #print("4.3")
    if B.LOWER:
        #print("4.3.1")
        if (F.LOWER and not F.HIGHER
            and F.topo_order > B.topo_order):
            return 'USE_BLUE'
        else:
            return 'USE_RED'
    if B.HIGHER:
        #print("4.3.2")
        if (F.HIGHER and not F.LOWER
            and F.topo_order < B.topo_order):
            return 'USE_BLUE'
        else:
            return 'USE_RED'
```

```

else:
    #print("4.3.3")
    if A.topo_order < B.topo_order:
        #print("4.3.3.1")
        if (alt_to_X == 'USE_BLUE'
            and alt_to_Y == 'USE_RED'):
            return 'USE_RED_OR_BLUE'
        if alt_to_X == 'USE_BLUE':
            return 'USE_BLUE'
        if alt_to_Y == 'USE_RED':
            return 'USE_RED'
        assert(False)
    else:
        #print("4.3.3.2")
        if (alt_to_X == 'USE_RED'
            and alt_to_Y == 'USE_BLUE'):
            return 'USE_RED_OR_BLUE'
        if alt_to_X == 'USE_RED':
            return 'USE_BLUE'
        if alt_to_Y == 'USE_BLUE':
            return 'USE_RED'
        assert(False)
assert(False)

def Compute_Primary_NHs_For_One_Src_To_Named_Proxy_Nodes(topo,src):
    for prefix in topo.named_proxy_dict:
        P = topo.named_proxy_dict[prefix]
        min_total_pref_cost = 2147483647
        for (adv_node, prefix_cost) in P.node_prefix_cost_list:
            total_pref_cost = (adv_node.primary_spf_metric
                               + prefix_cost)
            if total_pref_cost < min_total_pref_cost:
                min_total_pref_cost = total_pref_cost
                Copy_List_Items(P.primary_next_hops,
                               adv_node.primary_next_hops)
            elif total_pref_cost == min_total_pref_cost:
                for nh_intf in adv_node.primary_next_hops:
                    Add_Item_To_List_If_New(P.primary_next_hops,
                                             nh_intf)

def Select_Alts_For_One_Src_To_Named_Proxy_Nodes(topo,src):
    for prefix in topo.named_proxy_dict:
        P = topo.named_proxy_dict[prefix]
        P.alt_list = []
        for failed_intf in P.primary_next_hops:
            alt = Alternate()
            alt.failed_intf = failed_intf
            if failed_intf not in src.island_intf_list:

```

```

        alt.info = 'PRIM_NH_FOR_PROXY_NODE_NOT_IN_ISLAND'
    elif P.pnar1 is None:
        alt.info = 'NO_PNARs_EXIST_FOR_THIS_PREFIX'
    elif src is P.pnar1.node:
        alt.info = 'SRC_IS_PNAR'
    elif P.pnar2 is not None and src is P.pnar2.node:
        alt.info = 'SRC_IS_PNAR'
    elif P.pnar2 is None:
        #inherit alternates from the only pnar.
        alt.info = Select_Alternates(P.pnar1.node,
            failed_intf.remote_node, failed_intf)
    elif failed_intf in src.island_intf_list:
        alt.info = Select_Alternates_Proxy_Node(P,
            failed_intf.remote_node, failed_intf)

    if alt.info == 'USE_RED_OR_BLUE':
        alt.red_or_blue = \
            random.choice(['USE_RED', 'USE_BLUE'])
    if (alt.info == 'USE_BLUE'
        or alt.red_or_blue == 'USE_BLUE'):
        Copy_List_Items(alt.nh_list, P.blue_next_hops)
        alt.fec = 'BLUE'
        alt.prot = 'NODE_PROTECTION'
    elif (alt.info == 'USE_RED'
        or alt.red_or_blue == 'USE_RED'):
        Copy_List_Items(alt.nh_list, P.red_next_hops)
        alt.fec = 'RED'
        alt.prot = 'NODE_PROTECTION'
    elif (alt.info == 'PRIM_NH_IS_D_OR_OP_FOR_D'
        or alt.info == 'PRIM_NH_IS_OP_FOR_BOTH_X_AND_Y'):
        if failed_intf.OUTGOING and failed_intf.INCOMING:
            # cut-link: if there are parallel cut links, use
            # the link(s) with lowest metric that are not
            # primary intf or None
            cand_alt_list = [None]
            min_metric = 2147483647
            for intf in src.island_intf_list:
                if ( intf is not failed_intf and
                    (intf.remote_node is
                     failed_intf.remote_node)):
                    if intf.metric < min_metric:
                        cand_alt_list = [intf]
                        min_metric = intf.metric
                    elif intf.metric == min_metric:
                        cand_alt_list.append(intf)
            if cand_alt_list != [None]:
                alt.fec = 'GREEN'
                alt.prot = 'PARALLEL_CUTLINK'

```

```

        else:
            alt.fec = 'NO_ALTERNATE'
            alt.prot = 'NO_PROTECTION'
            Copy_List_Items(alt.nh_list, cand_alt_list)
    else:
        # set Z as the node to inherit blue next-hops from
        if alt.info == 'PRIM_NH_IS_D_OR_OP_FOR_D':
            Z = P.pnar1.node
        else:
            Z = P
        if failed_intf in Z.red_next_hops:
            Copy_List_Items(alt.nh_list, Z.blue_next_hops)
            alt.fec = 'BLUE'
            alt.prot = 'LINK_PROTECTION'
        else:
            assert(failed_intf in Z.blue_next_hops)
            Copy_List_Items(alt.nh_list, Z.red_next_hops)
            alt.fec = 'RED'
            alt.prot = 'LINK_PROTECTION'

elif alt.info == 'PRIM_NH_FOR_PROXY_NODE_NOT_IN_ISLAND':
    if (P.pnar2 == None and src is P.pnar1.node):
        #MRT Island is singly connected to non-island dest
        alt.fec = 'NO_ALTERNATE'
        alt.prot = 'NO_PROTECTION'
    elif P.node_id in src.blue_to_green_nh_dict:
        # blue to P goes to failed LFIN so use red to P
        Copy_List_Items(alt.nh_list, P.red_next_hops)
        alt.fec = 'RED'
        alt.prot = 'LINK_PROTECTION'
    elif P.node_id in src.red_to_green_nh_dict:
        # red to P goes to failed LFIN so use blue to P
        Copy_List_Items(alt.nh_list, P.blue_next_hops)
        alt.fec = 'BLUE'
        alt.prot = 'LINK_PROTECTION'
    else:
        Copy_List_Items(alt.nh_list, P.blue_next_hops)
        alt.fec = 'BLUE'
        alt.prot = 'LINK_PROTECTION'
elif alt.info == 'TEMP_NO_ALTERNATE':
    alt.fec = 'NO_ALTERNATE'
    alt.prot = 'NO_PROTECTION'

P.alt_list.append(alt)

def Run_Basic_MRT_for_One_Source(topo, src):
    MRT_Island_Identification(topo, src, 0, 0)
    Set_Island_Intf_and_Node_Lists(topo)

```

```
Set_GADAG_Root(topo,src)
Sort_Interfaces(topo)
Run_Lowpoint(topo)
Assign_Remaining_Lowpoint_Parents(topo)
Construct_GADAG_via_Lowpoint(topo)
Run_Assign_Block_ID(topo)
Add_Undirected_Links(topo)
Compute_MRT_NH_For_One_Src_To_Island_Dests(topo,src)
Store_MRT_Nexthops_For_One_Src_To_Island_Dests(topo,src)
Select_Alts_For_One_Src_To_Island_Dests(topo,src)
Store_Primary_and_Alts_For_One_Src_To_Island_Dests(topo,src)

def Store_GADAG_and_Named_Proxies_Once(topo):
    for node in topo.node_list:
        for intf in node.intf_list:
            if intf.OUTGOING:
                intf.SIMULATION_OUTGOING = True
            else:
                intf.SIMULATION_OUTGOING = False
    for prefix in topo.named_proxy_dict:
        P = topo.named_proxy_dict[prefix]
        topo.stored_named_proxy_dict[prefix] = P

def Run_Basic_MRT_for_All_Sources(topo):
    for src in topo.node_list:
        Reset_Computed_Node_and_Intf_Values(topo)
        Run_Basic_MRT_for_One_Source(topo,src)
    if src is topo.gadag_root:
        Store_GADAG_and_Named_Proxies_Once(topo)

def Run_MRT_for_One_Source(topo, src):
    MRT_Island_Identification(topo, src, 0, 0)
    Set_Island_Intf_and_Node_Lists(topo)
    Set_GADAG_Root(topo,src)
    Sort_Interfaces(topo)
    Run_Lowpoint(topo)
    Assign_Remaining_Lowpoint_Parents(topo)
    Construct_GADAG_via_Lowpoint(topo)
    Run_Assign_Block_ID(topo)
    Add_Undirected_Links(topo)
    Compute_MRT_NH_For_One_Src_To_Island_Dests(topo,src)
    Store_MRT_Nexthops_For_One_Src_To_Island_Dests(topo,src)
    Select_Alts_For_One_Src_To_Island_Dests(topo,src)
    Store_Primary_and_Alts_For_One_Src_To_Island_Dests(topo,src)
    Create_Basic_Named_Proxy_Nodes(topo)
    Attach_Named_Proxy_Nodes(topo)
    Compute_MRT_NHs_For_One_Src_To_Named_Proxy_Nodes(topo,src)
    Store_MRT_NHs_For_One_Src_To_Named_Proxy_Nodes(topo,src)
```

```

    Compute_Primary_NHs_For_One_Src_To_Named_Proxy_Nodes(topo,src)
    Store_Primary_NHs_For_One_Src_To_Named_Proxy_Nodes(topo,src)
    Select_Alts_For_One_Src_To_Named_Proxy_Nodes(topo,src)
    Store_Alts_For_One_Src_To_Named_Proxy_Nodes(topo,src)

def Run_Prim_SPF_for_One_Source(topo,src):
    Normal_SPF(topo, src)
    Store_Primary_NHs_For_One_Source_To_Nodes(topo,src)
    Create_Basic_Named_Proxy_Nodes(topo)
    Compute_Primary_NHs_For_One_Src_To_Named_Proxy_Nodes(topo,src)
    Store_Primary_NHs_For_One_Src_To_Named_Proxy_Nodes(topo,src)

def Run_MRT_for_All_Sources(topo):
    for src in topo.node_list:
        Reset_Computed_Node_and_Intf_Values(topo)
        if src in topo.island_node_list_for_test_gr:
            # src runs MRT if it is in same MRT island as test_gr
            Run_MRT_for_One_Source(topo,src)
            if src is topo.gadag_root:
                Store_GADAG_and_Named_Proxies_Once(topo)
        else:
            # src still runs SPF if not in MRT island
            Run_Prim_SPF_for_One_Source(topo,src)

def Write_Output_To_Files(topo,file_prefix):
    Write_GADAG_To_File(topo,file_prefix)
    Write_Both_MRTs_For_All_Dests_To_File(topo,file_prefix)
    Write_Alternates_For_All_Dests_To_File(topo,file_prefix)

def Create_Basic_Topology_Input_File(filename):
    data = [[01,02,10],[02,03,10],[03,04,11],[04,05,10,20],[05,06,10],
            [06,07,10],[06,07,10],[06,07,15],[07,01,10],[07,51,10],
            [51,52,10],[52,53,10],[53,03,10],[01,55,10],[55,06,10],
            [04,12,10],[12,13,10],[13,14,10],[14,15,10],[15,16,10],
            [16,17,10],[17,04,10],[05,76,10],[76,77,10],[77,78,10],
            [78,79,10],[79,77,10]]
    with open(filename + '.csv', 'w') as topo_file:
        for item in data:
            if len(item) > 3:
                line = (str(item[0])+','+','+str(item[1])+','+','+
                        str(item[2])+','+','+str(item[3])+'\n')
            else:
                line = (str(item[0])+','+','+str(item[1])+','+','+
                        str(item[2])+'\n')
            topo_file.write(line)

def Create_Complex_Topology_Input_File(filename):
    data = [[01,02,10],[02,03,10],[03,04,11],[04,05,10,20],[05,06,10],

```

```

        [06,07,10],[06,07,10],[06,07,15],[07,01,10],[07,51,10],
        [51,52,10],[52,53,10],[53,03,10],[01,55,10],[55,06,10],
        [04,12,10],[12,13,10],[13,14,10],[14,15,10],[15,16,10],
        [16,17,10],[17,04,10],[05,76,10],[76,77,10],[77,78,10],
        [78,79,10],[79,77,10]]
with open(filename + '.csv', 'w') as topo_file:
    for item in data:
        if len(item) > 3:
            line = (str(item[0])+','+str(item[1])+','+
                    str(item[2])+','+str(item[3]))+'\n')
        else:
            line = (str(item[0])+','+str(item[1])+','+
                    str(item[2]))+'\n')
        topo_file.write(line)

data = [[01,0],[02,0],[03,0],[04,0],[05,0],
        [06,0],[07,0],
        [51,0],[55,0],
        [12,0],[13,0],[14,0],[15,0],
        [16,0],[17,0],[76,0],[77,0],
        [78,0],[79,0]]
with open(filename + '.profile', 'w') as topo_file:
    for item in data:
        line = (str(item[0])+','+str(item[1]))+'\n')
        topo_file.write(line)

data = [[2001,05,100],[2001,07,120],[2001,03,130],
        [2002,13,100],[2002,15,110],
        [2003,52,100],[2003,78,100]]
with open(filename + '.prefix', 'w') as topo_file:
    for item in data:
        line = (str(item[0])+','+str(item[1])+','+
                str(item[2]))+'\n')
        topo_file.write(line)

def Generate_Basic_Topology_and_Run_MRT():
    this_gadag_root = 3
    Create_Basic_Topology_Input_File('basic_topo_input')
    topo = Create_Topology_From_File('basic_topo_input')
    res_file_base = 'basic_topo'
    Compute_Island_Node_List_For_Test_GR(topo, this_gadag_root)
    Raise_GADAG_Root_Selection_Priority(topo,this_gadag_root)
    Run_Basic_MRT_for_All_Sources(topo)
    Write_Output_To_Files(topo, res_file_base)

def Generate_Complex_Topology_and_Run_MRT():
    this_gadag_root = 3
    Create_Complex_Topology_Input_File('complex_topo_input')

```

```
topo = Create_Topology_From_File('complex_topo_input')
Add_Profile_IDs_from_File(topo,'complex_topo_input')
Add_Prefix_Advertisements_From_File(topo,'complex_topo_input')
Compute_Island_Node_List_For_Test_GR(topo, this_gadag_root)
Add_Prefixes_for_Non_Island_Nodes(topo)
res_file_base = 'complex_topo'
Raise_GADAG_Root_Selection_Priority(topo,this_gadag_root)
Run_MRT_for_All_Sources(topo)
Write_Output_To_Files(topo, res_file_base)
```

Generate_Basic_Topology_and_Run_MRT()

Generate_Complex_Topology_and_Run_MRT()

<CODE ENDS>

Appendix B. Constructing a GADAG using SPF's

The basic idea in this method for constructing a GADAG is to use slightly-modified SPF computations to find ears. In every block, an SPF computation is first done to find a cycle from the local root and then SPF computations in that block find ears until there are no more interfaces to be explored. The used result from the SPF computation is the path of interfaces indicated by following the previous hops from the minimized IN_GADAG node back to the SPF root.

To do this, first all cut-vertices must be identified and local-roots assigned as specified in Figure 12.

The slight modifications to the SPF are as follows. The root of the block is referred to as the block-root; it is either the GADAG root or a cut-vertex.

- a. The SPF is rooted at a neighbor *x* of an IN_GADAG node *y*. All links between *y* and *x* are marked as TEMP_UNUSABLE. They should not be used during the SPF computation.
- b. If *y* is not the block-root, then it is marked TEMP_UNUSABLE. It should not be used during the SPF computation. This prevents ears from starting and ending at the same node and avoids cycles; the exception is because cycles to/from the block-root are acceptable and expected.
- c. Do not explore links to nodes whose local-root is not the block-root. This keeps the SPF confined to the particular block.
- d. Terminate when the first IN_GADAG node *z* is minimized.

- e. Respect the existing directions (e.g. INCOMING, OUTGOING, UNDIRECTED) already specified for each interface.

```

Mod_SPF(spf_root, block_root)
  Initialize spf_heap to empty
  Initialize nodes' spf_metric to infinity
  spf_root.spf_metric = 0
  insert(spf_heap, spf_root)
  found_in_gadag = false
  while (spf_heap is not empty) and (found_in_gadag is false)
    min_node = remove_lowest(spf_heap)
    if min_node.IN_GADAG
      found_in_gadag = true
    else
      foreach interface intf of min_node
        if ((intf.OUTGOING or intf.UNDIRECTED) and
            ((intf.remote_node.localroot is block_root) or
             (intf.remote_node is block_root)) and
            (intf.remote_node is not TEMP_UNUSABLE) and
            (intf is not TEMP_UNUSABLE))
          path_metric = min_node.spf_metric + intf.metric
          if path_metric < intf.remote_node.spf_metric
            intf.remote_node.spf_metric = path_metric
            intf.remote_node.spf_prev_intf = intf
            insert_or_update(spf_heap, intf.remote_node)
  return min_node

```

```

SPF_for_Ear(cand_intf.local_node, cand_intf.remote_node, block_root,
method)
  Mark all interfaces between cand_intf.remote_node
  and cand_intf.local_node as TEMP_UNUSABLE
  if cand_intf.local_node is not block_root
    Mark cand_intf.local_node as TEMP_UNUSABLE
  Initialize ear_list to empty
  end_ear = Mod_SPF(spf_root, block_root)
  y = end_ear.spf_prev_hop
  while y.local_node is not spf_root
    add_to_list_start(ear_list, y)
    y.local_node.IN_GADAG = true
    y = y.local_node.spf_prev_intf
  if(method is not hybrid)
    Set_Ear_Direction(ear_list, cand_intf.local_node,
end_ear, block_root)
  Clear TEMP_UNUSABLE from all interfaces between
  cand_intf.remote_node and cand_intf.local_node

```

```
Clear TEMP_UNUSABLE from cand_intf.local_node
return end_ear
```

Figure 31: Modified SPF for GADAG construction

Assume that an ear is found by going from y to x and then running an SPF that terminates by minimizing z (e.g. $y \leftarrow x \dots q \leftarrow z$). Now it is necessary to determine the direction of the ear; if $y \ll z$, then the path should be $y \rightarrow x \dots q \rightarrow z$ but if $y \gg z$, then the path should be $y \leftarrow x \dots q \leftarrow z$. In Section 5.5, the same problem was handled by finding all ears that started at a node before looking at ears starting at nodes higher in the partial order. In this GADAG construction method, using that approach could mean that new ears aren't added in order of their total cost since all ears connected to a node would need to be found before additional nodes could be found.

The alternative is to track the order relationship of each node with respect to every other node. This can be accomplished by maintaining two sets of nodes at each node. The first set, `Higher_Nodes`, contains all nodes that are known to be ordered above the node. The second set, `Lower_Nodes`, contains all nodes that are known to be ordered below the node. This is the approach used in this GADAG construction method.

```

Set_Ear_Direction(ear_list, end_a, end_b, block_root)
  // Default of A_TO_B for the following cases:
  // (a) end_a and end_b are the same (root)
  // or (b) end_a is in end_b's Lower Nodes
  // or (c) end_a and end_b were unordered with respect to each
  //      other
  direction = A_TO_B
  if (end_b is block_root) and (end_a is not end_b)
    direction = B_TO_A
  else if end_a is in end_b.Higher_Nodes
    direction = B_TO_A
  if direction is B_TO_A
    foreach interface i in ear_list
      i.UNDIRECTED = false
      i.INCOMING = true
      i.remote_intf.UNDIRECTED = false
      i.remote_intf.OUTGOING = true
  else
    foreach interface i in ear_list
      i.UNDIRECTED = false
      i.OUTGOING = true
      i.remote_intf.UNDIRECTED = false
      i.remote_intf.INCOMING = true
  if end_a is end_b
    return
  // Next, update all nodes' Lower_Nodes and Higher_Nodes
  if (end_a is in end_b.Higher_Nodes)
    foreach node x where x.localroot is block_root
      if end_a is in x.Lower_Nodes
        foreach interface i in ear_list
          add i.remote_node to x.Lower_Nodes
      if end_b is in x.Higher_Nodes
        foreach interface i in ear_list
          add i.local_node to x.Higher_Nodes
  else
    foreach node x where x.localroot is block_root
      if end_b is in x.Lower_Nodes
        foreach interface i in ear_list
          add i.local_node to x.Lower_Nodes
      if end_a is in x.Higher_Nodes
        foreach interface i in ear_list
          add i.remote_node to x.Higher_Nodes

```

Figure 32: Algorithm to assign links of an ear direction

A goal of this GADAG construction method is to find the shortest cycles and ears. An ear is started by going to a neighbor x of an IN_GADAG node y . The path from x to an IN_GADAG node is minimal,

since it is computed via SPF. Since a shortest path is made of shortest paths, to find the shortest ears requires reaching from the set of IN_GADAG nodes to the closest node that isn't IN_GADAG. Therefore, an ordered tree is maintained of interfaces that could be explored from the IN_GADAG nodes. The interfaces are ordered by their characteristics of metric, local loopback address, remote loopback address, and ifindex, based on the Interface_Compare function defined in Figure 14.

This GADAG construction method ignores interfaces picked from the ordered list that belong to the block root if the block in which the interface is present already has an ear that has been computed. This is necessary since we allow at most one incoming interface to a block root in each block. This requirement stems from the way next-hops are computed as was seen in Section 5.7. After any ear gets computed, we traverse the newly added nodes to the GADAG and insert interfaces whose far end is not yet on the GADAG to the ordered tree for later processing.

Finally, cut-links are a special case because there is no point in doing an SPF on a block of 2 nodes. The algorithm identifies cut-links simply as links where both ends of the link are cut-vertices. Cut-links can simply be added to the GADAG with both OUTGOING and INCOMING specified on their interfaces.

```

add_eligible_interfaces_of_node(ordered_intfs_tree,node)
  for each interface of node
    if intf.remote_node.IN_GADAG is false
      insert(intf,ordered_intfs_tree)

check_if_block_has_ear(x,block_id)
  block_has_ear = false
  for all interfaces of x
    if ( (intf.remote_node.block_id == block_id) &&
        intf.remote_node.IN_GADAG )
      block_has_ear = true
return block_has_ear

Construct_GADAG_via_SPF(topology, root)
  Compute_Localroot (root,root)
  Assign_Block_ID(root,0)
  root.IN_GADAG = true
  add_eligible_interfaces_of_node(ordered_intfs_tree,root)
  while ordered_intfs_tree is not empty
    cand_intf = remove_lowest(ordered_intfs_tree)
    if cand_intf.remote_node.IN_GADAG is false
      if L(cand_intf.remote_node) == D(cand_intf.remote_node)
        // Special case for cut-links

```

```

        cand_intf.UNDIRECTED = false
        cand_intf.remote_intf.UNDIRECTED = false
        cand_intf.OUTGOING = true
        cand_intf.INCOMING = true
        cand_intf.remote_intf.OUTGOING = true
        cand_intf.remote_intf.INCOMING = true
        cand_intf.remote_node.IN_GADAG = true
        add_eligible_interfaces_of_node(
            ordered_intfs_tree, cand_intf.remote_node)
    else
        if (cand_intf.remote_node.local_root ==
            cand_intf.local_node) &&
            check_if_block_has_ear(cand_intf.local_node,
                                   cand_intf.remote_node.block_id))
            /* Skip the interface since the block root
               already has an incoming interface in the
               block */
        else
            ear_end = SPF_for_Ear(cand_intf.local_node,
                                   cand_intf.remote_node,
                                   cand_intf.remote_node.localroot,
                                   SPF method)
            y = ear_end.spf_prev_hop
            while y.local_node is not cand_intf.local_node
                add_eligible_interfaces_of_node(
                    ordered_intfs_tree, y.local_node)
                y = y.local_node.spf_prev_intf

```

Figure 33: SPF-based method for GADAG construction

Appendix C. Constructing a GADAG using a hybrid method

The idea of this method is to combine the salient features of the lowpoint inheritance and SPF methods. To this end, we process nodes as they get added to the GADAG just like in the lowpoint inheritance by maintaining a stack of nodes. This ensures that we do not need to maintain lower and higher sets at each node to ascertain ear directions since the ears will always be directed from the node being processed towards the end of the ear. To compute the ear however, we resort to an SPF to have the possibility of better ears (path lengths) thus giving more flexibility than the restricted use of lowpoint/dfs parents.

Regarding ears involving a block root, unlike the SPF method which ignored interfaces of the block root after the first ear, in the hybrid method we would have to process all interfaces of the block root before moving on to other nodes in the block since the direction

of an ear is pre-determined. Thus, whenever the block already has an ear computed, and we are processing an interface of the block root, we mark the block root as unusable before the SPF run that computes the ear. This ensures that the SPF terminates at some node other than the block-root. This in turn guarantees that the block-root has only one incoming interface in each block, which is necessary for correctly computing the next-hops on the GADAG.

As in the SPF gadag, bridge ears are handled as a special case.

The entire algorithm is shown below in Figure 34

```

find_spf_stack_ear(stack, x, y, xy_intf, block_root)
  if L(y) == D(y)
    // Special case for cut-links
    xy_intf.UNDIRECTED = false
    xy_intf.remote_intf.UNDIRECTED = false
    xy_intf.OUTGOING = true
    xy_intf.INCOMING = true
    xy_intf.remote_intf.OUTGOING = true
    xy_intf.remote_intf.INCOMING = true
    xy_intf.remote_node.IN_GADAG = true
    push y onto stack
    return
  else
    if (y.local_root == x) &&
      check_if_block_has_ear(x,y.block_id)
      //Avoid the block root during the SPF
      Mark x as TEMP_UNUSABLE
    end_ear = SPF_for_Ear(x,y,block_root,hybrid)
    If x was set as TEMP_UNUSABLE, clear it
    cur = end_ear
    while (cur != y)
      intf = cur.spf_prev_hop
      prev = intf.local_node
      intf.UNDIRECTED = false
      intf.remote_intf.UNDIRECTED = false
      intf.OUTGOING = true
      intf.remote_intf.INCOMING = true
      push prev onto stack
      cur = prev
    xy_intf.UNDIRECTED = false
    xy_intf.remote_intf.UNDIRECTED = false
    xy_intf.OUTGOING = true
    xy_intf.remote_intf.INCOMING = true
    return

Construct_GADAG_via_hybrid(topology,root)

```

```
Compute_Localroot (root,root)
Assign_Block_ID(root,0)
root.IN_GADAG = true
Initialize Stack to empty
push root onto Stack
while (Stack is not empty)
  x = pop(Stack)
  for each interface intf of x
    y = intf.remote_node
    if y.IN_GADAG is false
      find_spf_stack_ear(stack, x, y, intf, y.block_root)
```

Figure 34: Hybrid GADAG construction method

Authors' Addresses

Gabor Sandor Enyedi
Ericsson
Konyves Kalman krt 11
Budapest 1097
Hungary

Email: Gabor.Sandor.Enyedi@ericsson.com

Andras Csaszar
Ericsson
Konyves Kalman krt 11
Budapest 1097
Hungary

Email: Andras.Csaszar@ericsson.com

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

Email: akatlas@juniper.net

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

Email: cbowers@juniper.net

Abishek Gopalan
University of Arizona
1230 E Speedway Blvd.
Tucson, AZ 85721
USA

Email: abishek@ece.arizona.edu

INTERNET-DRAFT
Intended status: Proposed Standard

Z. Li
S. Zhuang
G. Yan
D. Eastlake
Huawei
November 8, 2018

Expires: May 7, 2019

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

Table of Contents

1. Introduction.....	3
2. Definitions and Acronyms.....	3
3. Introduction.....	4
3.1 Tunnel Policy.....	4
3.1.1 Selection Sequence.....	4
3.1.2 Tunnel Binding.....	4
3.2 Tunnel Selector for Routes.....	5
3.3 Tunnel Selector for VPNs.....	6
4. Design of Data Model.....	7
4.1 Tunnel Policy YANG Model.....	7
4.2 Tunnel Selector YANG Model.....	7
5. Tunnel Policy Yang Module.....	9
6. IANA Considerations.....	24
7. Security Considerations.....	24
Acknowledgements.....	25
Informational References.....	26
Normative References.....	26
Authors' Addresses.....	27

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:

- o Selection Sequence: The system selects a tunnel for the service based on the tunnel type priorities defined in the tunnel policy.
- o 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?     tnlnbaseTnlPolicyType
|   |   +--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?   tnlnbaseSelTnlType
|   |   |   |   +--rw selTnlType2?   tnlnbaseSelTnlType
|   |   |   |   +--rw selTnlType3?   tnlnbaseSelTnlType
|   |   |   |   +--rw selTnlType4?   tnlnbaseSelTnlType
|   |   |   |   +--rw selTnlType5?   tnlnbaseSelTnlType
|   |   |   |   +--rw selTnlType6?   tnlnbaseSelTnlType
|   |   |   |   +--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 "invaield" {
        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 tnImbaseTnlPolicyType;
    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
```



```
        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 tnldbbaseSelTnlType;
    default "invaile";
    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='invaile' and "
        + "..../tnlPolicyType='tnlSelectSeq' or "
        + "..selTnlType1='invaile')";
    type tnldbbaseSelTnlType;
    default "invaile";
    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='invaile' or "
        + "..selTnlType2='invaile')";
    type tnldbbaseSelTnlType;
    default "invaile";
    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='invaile' or "
        + "..selTnlType2='invaile' or "
        + "..selTnlType3='invaile')";
    type tnldbbaseSelTnlType;
    default "invaile";
    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='invaield' or "
            + "../selTnlType2='invaield' or "
            + "../selTnlType3='invaield' or "
            + "../selTnlType4='invaield')";
        type tnImbaseSelTnlType;
        default "invaield";
        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='invaield' or "
            + "../selTnlType2='invaield' or "
            + "../selTnlType3='invaield' or "
            + "../selTnlType4='invaield' or "
            + "../selTnlType5='invaield')";
        type tnImbaseSelTnlType;
        default "invaield";
        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

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

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License. The definitive version of an IETF Document is that published by, or under the auspices of, the IETF. Versions of IETF Documents that are published by third parties, including those that are translated into other languages, should not be considered to be definitive versions of IETF Documents. The definitive version of these Legal Provisions is that published by, or under the auspices of, the IETF. Versions of these Legal Provisions that are published by third parties, including those that are translated into other languages, should not be considered to be definitive versions of these Legal Provisions. For the avoidance of doubt, each Contributor to the IETF Standards Process licenses each Contribution that he or she makes as part of the IETF Standards Process to the IETF Trust pursuant to the provisions of RFC 5378. No language to the contrary, or terms, conditions or rights that differ from or are inconsistent with the rights and licenses granted under RFC 5378, shall have any effect and shall be null and void, whether published or posted by such Contributor, or included with or in such Contribution.

Routing Area Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 5, 2015

S. Litkowski
Orange Business Service
March 4, 2015

Link State protocols SPF trigger and delay algorithm impact on IGP
microloops
draft-litkowski-rtgwg-spf-uloop-pb-statement-02

Abstract

A micro-loop is a packet forwarding loop that may occur transiently among two or more routers in a hop-by-hop packet forwarding paradigm.

In this document, we are trying to analyze the impact of using different Link State IGP implementations in a single network in regards of microloops. The analysis is focused on the SPF triggers and SPF delay algorithm.

Requirements Language

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

Status of This Memo

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

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

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

This Internet-Draft will expire on September 5, 2015.

Copyright Notice

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

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

Table of Contents

1. Introduction	2
2. Problem statement	3
3. SPF trigger strategies	4
4. SPF delay strategies	5
4.1. Two step SPF delay	5
4.2. Exponential backoff	6
5. Mixing strategies	7
6. Proposed work items	11
7. Security Considerations	13
8. Acknowledgements	13
9. IANA Considerations	13
10. Normative References	13
Author's Address	13

1. Introduction

Link State IGP protocols are based on a topology database on which a SPF (Shortest Path First) algorithm like Dijkstra is implemented to find the optimal routing paths.

Specifications like IS-IS ([RFC1195]) propose some optimization of the route computation (See Appendix C.1) but not all the implementations are following those not mandatory optimizations.

We will call SPF trigger, the events that would lead to a new SPF computation based on the topology.

Link State IGP protocols, like OSPF ([RFC2328]) and IS-IS ([RFC1195]), are using plenty of timers to control the router behavior in case of churn : SPF delay, PRC delay, LSP generation delay, LSP flooding delay, LSP retransmission interval ...

Some of those timers are standardized in protocol specification, some are not especially the SPF computation related timers.

For non standardized timers, implementations are free to implement it in any way. For some standardized timer, we can also see that rather than using static configurable values for such timer, implementations may offer dynamically adjusted timers to help controlling the churn.

We will call SPF delay, the delay timer that exists in most implementations that makes codes to wait before running SPF computation after a SPF trigger is received.

A micro-loop is a packet forwarding loop that may occur transiently among two or more routers in a hop-by-hop packet forwarding paradigm. We can observe that these micro-loops are formed when two routers do not update their Forwarding Information Base (FIB) for a certain prefix at the same time. The micro-loop phenomenon is described in [I-D.ietf-rtgwg-microloop-analysis].

Routers have more and more powerful controlplane and dataplane that reduce the Control plane to Forwarding plane overhead during the convergence process. Even if FIB update is still reasonably the highest contributor in the convergence time for large network, its duration is reducing more and more and may become comparable to protocol timers. This is particular true in small and medium networks.

In multi vendor networks, using different implementations of a link state protocol may favor micro-loops creation during convergence time due to deprecancies of timers. Service Providers are already aware to use similar timers for all the network as best practice, but sometimes it is not possible due to limitation of implementations.

This document will present why it sounds important for service provider to have consistent implementations of Link State protocols across vendors. We are particularly analyzing the impact of using different Link State IGP implementations in a single network in regards of microloops. The analysis is focused on the SPF triggers and SPF delay algorithm in a first step.

This document is only stating the problem, and defining some work items but its not intended to provide a solution.

2. Problem statement

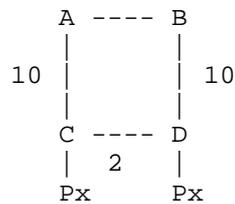


Figure 1

In the figure above, A uses primarily the AC link to reach C. When the AC link fails, IGP convergence occurs. If A converges before B, A will forward traffic to C through B, but as B has not converged yet, B will loop back traffic to A, leading to a microloop.

The micro-loop appears due to the asynchronous convergence of nodes in a network when an event occurs.

Multiple factors (and combination of these factors) may increase the probability for a micro-loop to appear :

- o delay of failure notification : the more B is advised of the failure later than A, the more a micro-loop may appear.
- o SPF delay : most of the implementations supports a delay for the SPF computation to try to catch as many events as possible. If A uses a SPF delay timer of x msec and B uses a SPF delay timer of y msec and $x < y$, B would start converging after A leading to a potential microloop.
- o SPF computation time : mostly a matter of CPU power and optimizations like incremental SPF. If A computes SPF faster than B, there is a chance for a microloop to appear. CPUs are today faster enough to consider SPF computation time as negligible (order of msec in a large network).
- o RIB and FIB prefix insertion speed or ordering : highly implementation dependant.

This document will focus on analysis SPF delay (and associated triggers).

3. SPF trigger strategies

Depending of the change advertised in LSP/LSA, the topology may be affected or not. An implementation can decide to not run SPF (and only run IP reachability) if the advertised change is not affecting topology.

Different strategies exists to trigger SPF :

1. Always run full SPF whatever the change to process.
2. Run only Full SPF when required : e.g. if a link fails, a local node will run an SPF for its local LSP update. If the LSP from the neighbor (describing the same failure) is received after SPF has started, the local node can decide that a new full SPF is not required as the topology has not change.
3. If topology does not change, only recompute reachability.

As pointed in Section 1, SPF optimization are not mandatory in specifications, leading to multiple strategies to be implemented.

4. SPF delay strategies

Implementations of link state routing protocols use different strategies to delay SPF :

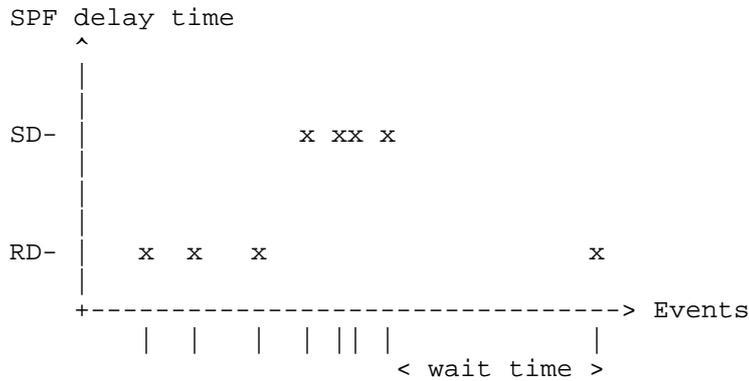
1. Two steps.
2. Exponential backoff.

4.1. Two step SPF delay

The SPF delay is managed by four parameters :

- o Rapid delay : amount of time to wait before running SPF.
- o Rapid runs : amount of consecutive SPF runs that can run using rapid delay. When amount is exceeded router moves to slow delay.
- o Slow delay : amount of time to wait before running SPF.
- o Wait time : amount of time to wait without events before going back to rapid delay.

Example : Rapid delay = 50msec, Rapid runs = 3, Slow delay = 1sec, Wait time = 2sec

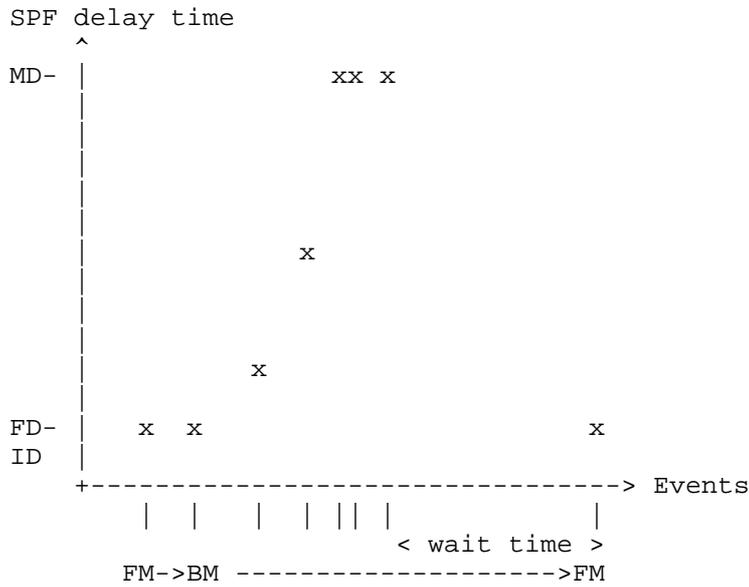


4.2. Exponential backoff

The algorithm has two mode : fast mode and backoff mode. In backoff mode, the SPF delay is increasing exponentially at each run. The SPF delay is managed by four parameters :

- o First delay : amount of time to wait before running SPF. This delay is used on when SPF is in fast mode.
- o Incremental delay : amount of time to wait before running SPF. This delay is used on when SPF is in backoff mode and increments exponentially at each SPF run.
- o Maximum delay : maximum amount of time to wait before running SPF.
- o Wait time : amount of time to wait without events before going back to fast mode.

Example : First delay = 50msec, Incremental delay = 50msec, Maximum delay = 1sec, Wait time = 2sec



5. Mixing strategies

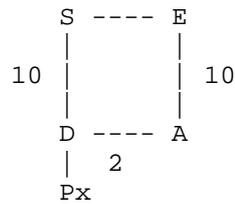
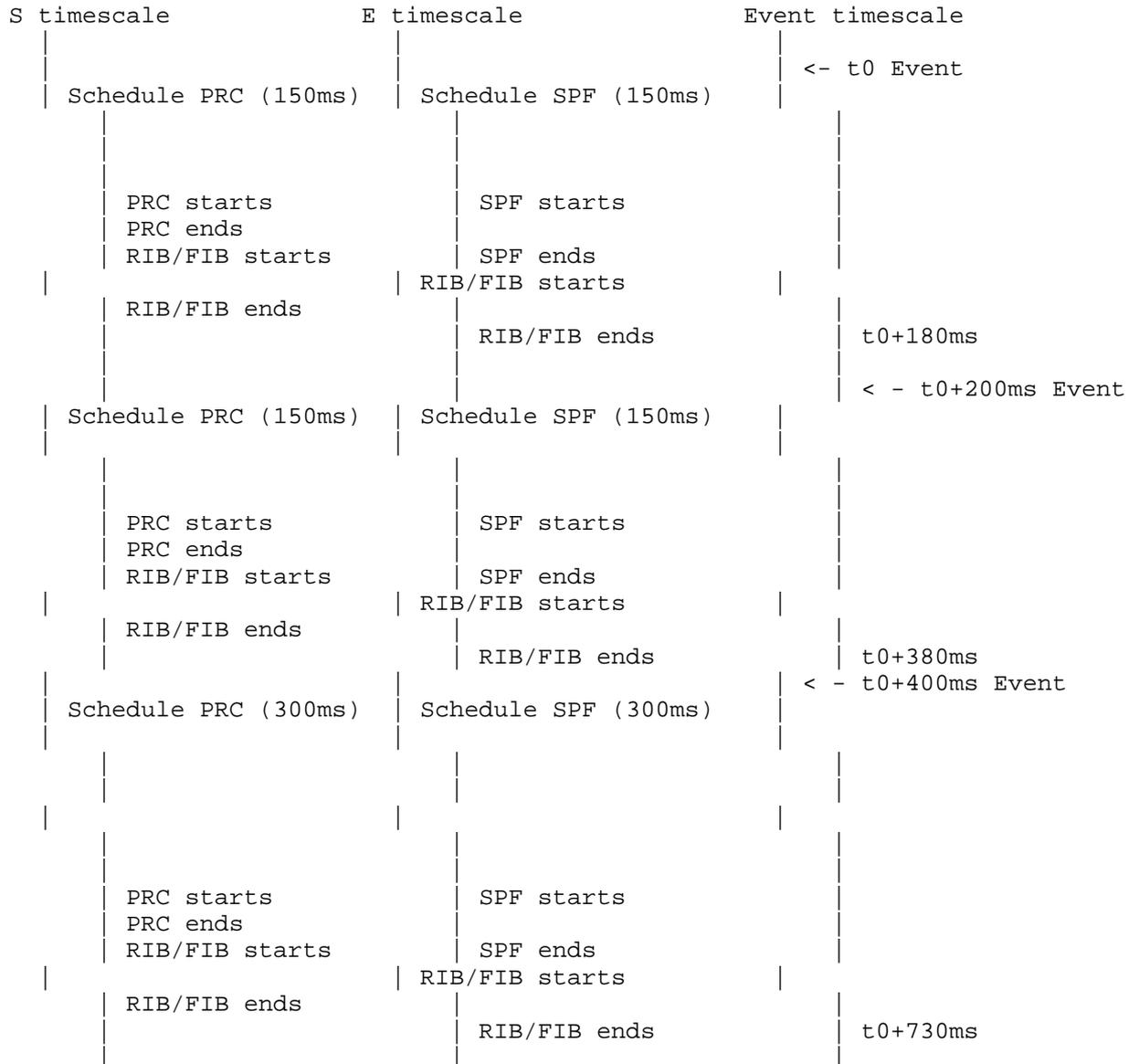


Figure 2

In the diagram above, we consider a flow of packet from S to D. We consider that S is using optimized SPF triggering (Full SPF is triggered only when necessary), and two steps SPF delay (rapid=150ms,rapid-runs=3, slow=1s). As implementation of S is optimized, Partial Reachability Computation (PRC) is available. We consider the same timers as SPF for delaying PRC. We consider that E is using a SPF trigger strategy that always compute Full SPF and exponential backoff strategy for SPF delay (start=150ms, inc=150ms, max=1s)

We also consider the following sequence of events (note : the timescale does not intend to represent a real router timescale where jitters are introduced to all timers) :

- o t0 : a prefix is declared down in the network.
- o t0+200ms : the prefix is declared as up.
- o t0+400ms : a prefix is declared down in the network.
- o t0+1000ms : S-D link fails.



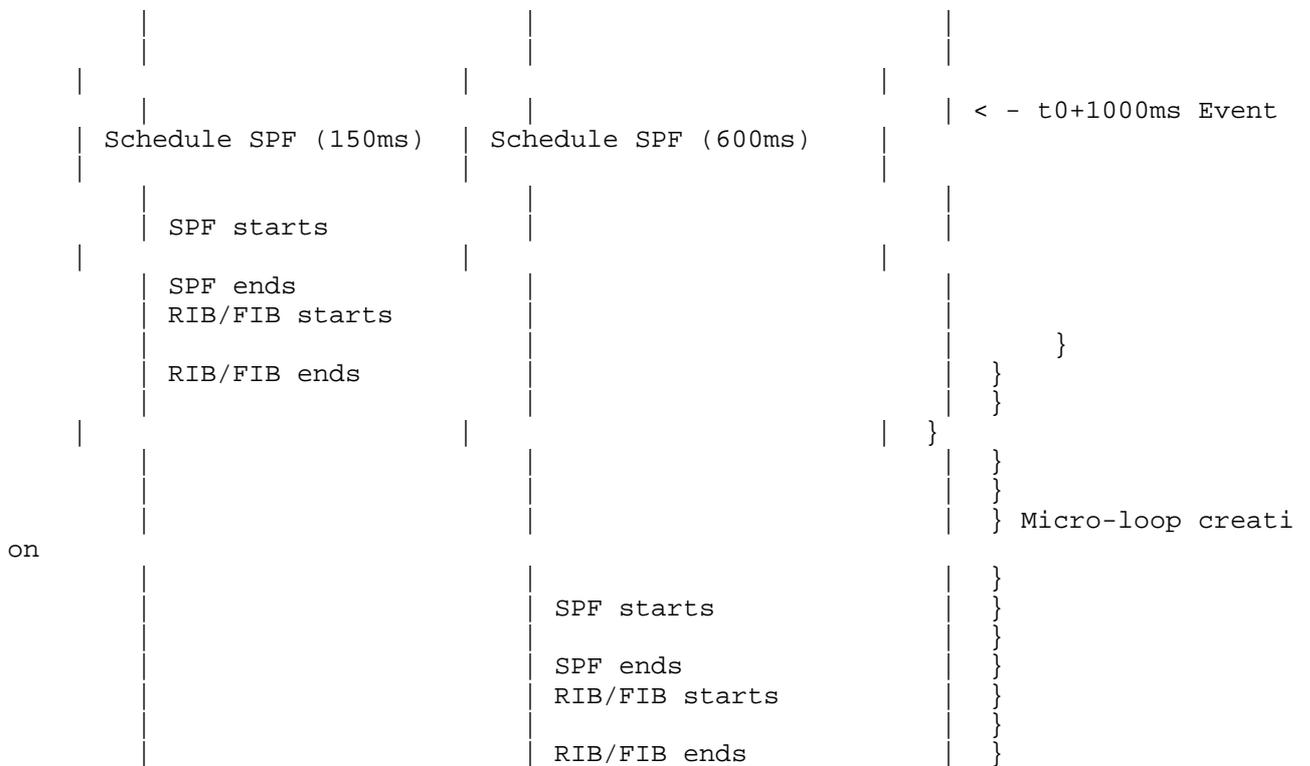
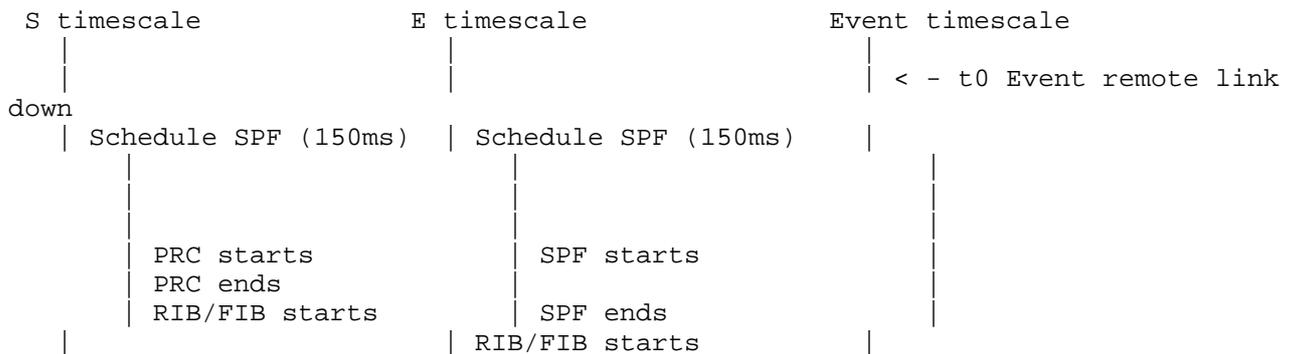


Figure 3

In the figure above, we can see that due to deprecancies in SPF management, after multiple events (different types of event), SPF delays are completely misaligned between nodes leading to long microloop creation.

The same issue can also appear with only single type of events as displayed below :





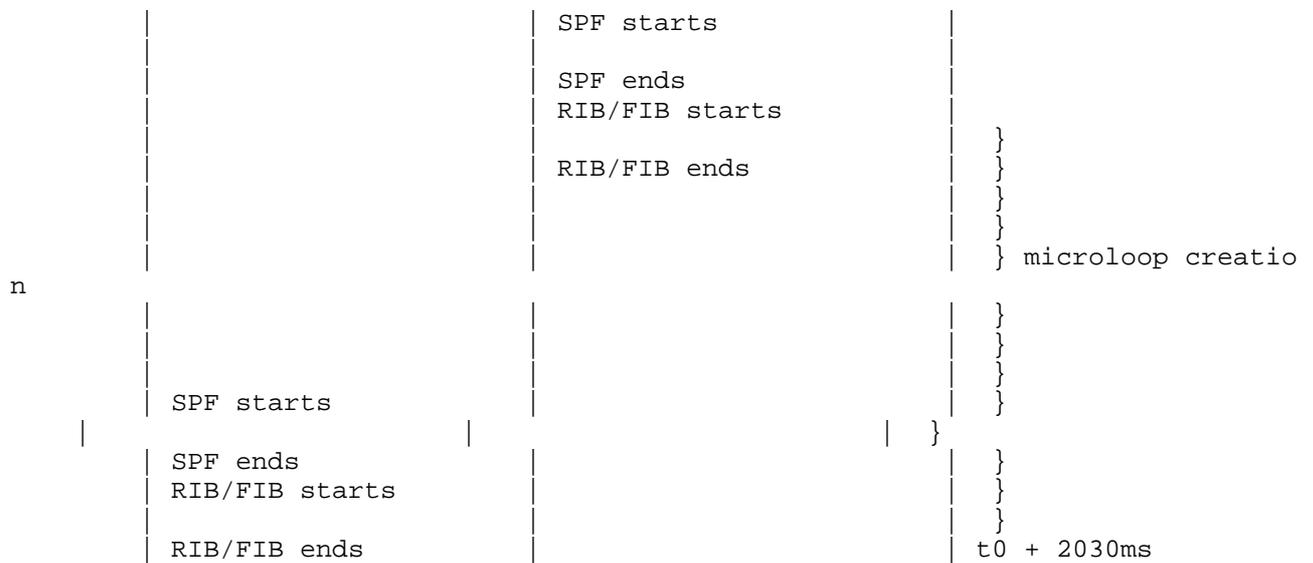


Figure 4

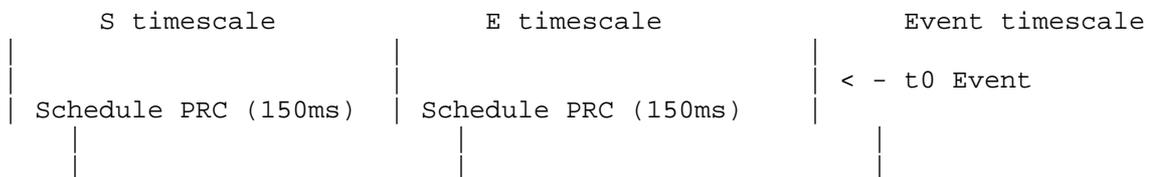
6. Proposed work items

In order to enhance the current LinkState IGP behavior, authors would encourage working on standardization of some behaviors.

Authors are proposing the following work items :

- o Standardize SPF trigger strategy.
- o Standardize computation timer scope : single timer for all computation operations, separated timers ...
- o Standardize "slowdown" timer algorithm including its association to a particular timer : authors of this document does not presume that the same algorithm must be used for all timers.

Using the same event sequence as in figure 2, we may expect fewer and/or shorter microloops using standardized implementations.



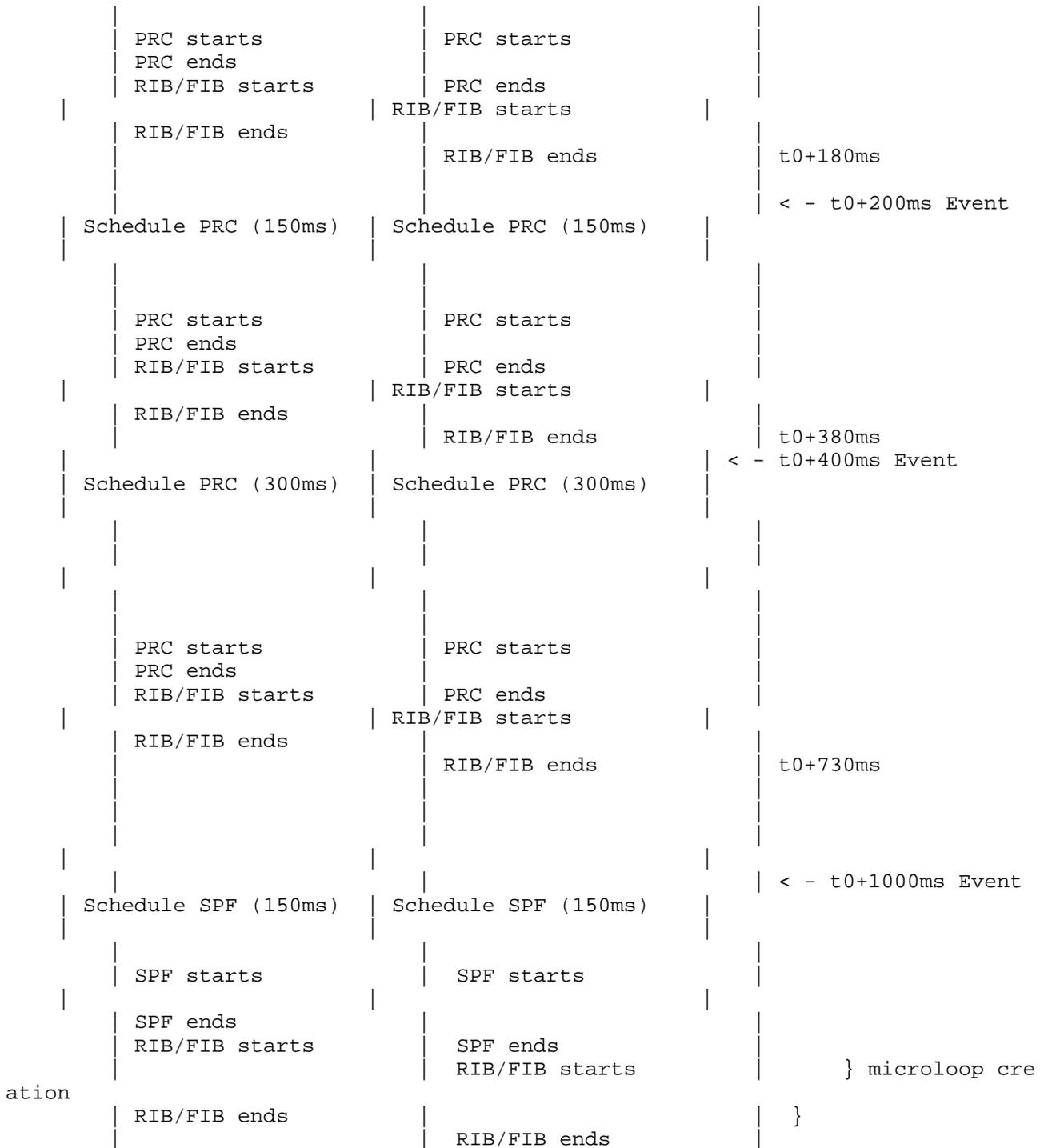




Figure 5

As displayed above, there could be some other parameters like router computation power, flooding timers that may also influence microloops. In the figure 5, we consider E to be a bit slower than S, leading to microloop creation. Despite of this, we expect that by aligning implementations at least on SPF trigger and SPF delay, service provider may reduce number or duration of microloops.

7. Security Considerations

This document does not introduce any security consideration.

8. Acknowledgements

9. IANA Considerations

This document has no action for IANA.

10. Normative References

[I-D.ietf-rtgwg-microloop-analysis]

Zinin, A., "Analysis and Minimization of Microloops in Link-state Routing Protocols", draft-ietf-rtgwg-microloop-analysis-01 (work in progress), October 2005.

[RFC1195] Callon, R., "Use of OSI IS-IS for routing in TCP/IP and dual environments", RFC 1195, December 1990.

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

[RFC2328] Moy, J., "OSPF Version 2", STD 54, RFC 2328, April 1998.

Author's Address

Stephane Litkowski
Orange Business Service

Email: stephane.litkowski@orange.com

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/lid-abstracts.txt>

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

This Internet-Draft will expire on October 5, 2016.

Copyright Notice

Copyright (c) 2016 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction.....	2
1.1. Terminology.....	2
2. VRRP YANG model overview.....	3
3. VRRP YANG module.....	7
4. Security Considerations.....	28
5. Contributors.....	28
6. References.....	29
6.1. Normative References.....	29
6.2. Informative References.....	29

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:

- o augment

- o data model
- o 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 {
```


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

q

Network Working Group
Internet-Draft
Intended status: Informational
Expires: April 21, 2016

J. George
Google
L. Fang
Microsoft
E. Osborne
Level 3
R. Shakir
Jive Communications
October 19, 2015

MPLS / TE Model for Service Provider Networks
draft-openconfig-mpls-consolidated-model-02

Abstract

This document defines a framework for a YANG data model for configuring and managing label switched paths, including the signaling protocols, traffic engineering, and operational aspects based on carrier and content provider operational requirements.

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.

Copyright Notice

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

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

to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1.	Introduction	2
1.1.	Goals and approach	2
2.	Model overview	4
2.1.	MPLS global	5
2.2.	TE global attributes	5
2.3.	TE interface attributes overview	6
2.4.	Signaling protocol overview	7
2.5.	LSP overview	8
3.	Example use cases	11
3.1.	Traffic engineered p2p LSP signaled with RSVP	11
3.2.	Traffic engineered LSP signaled with SR	12
3.3.	IGP-congruent LDP-signaled LSP	13
4.	Security Considerations	14
5.	IANA Considerations	14
6.	YANG modules	14
6.1.	MPLS base modules	15
6.2.	MPLS LSP submodules	30
6.3.	MPLS signaling protocol modules	50
7.	Contributing Authors	83
8.	Acknowledgements	84
9.	References	84
	Authors' Addresses	85

1. Introduction

This document describes a YANG [RFC6020] data model for MPLS and traffic engineering, covering label switched path (LSP) configuration, as well as signaling protocol configuration. The model is intended to be vendor-neutral, in order to allow operators to manage MPLS in heterogeneous environments with physical or virtual devices (routers, switches, servers, etc.) supplied by multiple vendors. The model is also intended to be readily mapped to existing implementations, to facilitate support from as large a set of routing hardware and software vendors as possible.

1.1. Goals and approach

The focus area of the model in this revision, is to set forth a framework for MPLS, with hooks into which information specific to various signaling-protocols can be added. The framework is built around functionality from a network operator perspective rather than

a signaling protocol-centric approach. For example, a traffic-engineered LSP will have configuration relating to its path computation method, regardless of whether it is signaled with RSVP-TE or with segment routing. Thus, rather than creating separate per-signaling protocol models and trying to stitch them under a common umbrella, this framework focuses on functionality, and adds signaling protocol-specific information under it where applicable.

This model does not aim to be feature complete (i.e., cover all possible aspects or features of MPLS). Rather its development is driven by examination of actual production configurations in use across a number of operator network deployments.

Configuration items 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 companion modules that augment the current model. This allows clarity in identifying data that is part of the vendor-neutral model.

An important aspect of the model is the representation of operational state data. This draft takes the approach described in [I-D.openconfig-netmod-opstate] and models configuration and operational state together. Thus, rather than building a separate tree of operational state, the operational state and configuration data are located in parallel containers at the leaves of the data model. This approach allows easy reuse of groupings across models, as well as making it easier to correlate configuration and state.

The consolidated MPLS model encompasses the signaling protocols, label-switched paths (configuration and operational state), and generic TE attributes. The model is designed from an operational and functional perspective, rather than focusing on protocol-centric configuration. This allows protocol-independent functions to be logically separated from protocol-specific details.

One question that arises in this approach is how the consolidated model is integrated with routing instances (e.g., VRFs). This model should be considered as part of a higher level network device model which includes definitions for other routing protocols and system services. For example, in [I-D.openconfig-netmod-model-structure], VRFs and other logical instances are defined with MPLS/TE components within VRFs as appropriate. In particular, some parts of the MPLS model would be instantiated within a VRF, while other parts would have common definitions across VRFs.

Where possible, naming in the model follows conventions used in available standards documents, and otherwise tries to be self-explanatory with sufficient descriptions of the intended behavior. Similarly, configuration data value constraints and default values, where used, are based on recommendations in current standards documentation. Since implementations vary widely in this respect, this version of the model specifies only a limited set of defaults and ranges with the expectation of being more prescriptive in future versions based on actual operator use.

Note that this version of the model is a work-in-progress in several respects. Although we present a complete framework for MPLS and traffic engineering from an operational perspective, some signaling protocol configuration will be completed in future revisions. The current revision has focus on traffic engineered LSPs signaled with RSVP.

2. Model overview

The overall MPLS model is defined across several YANG modules and submodules but at a high level is organized into 4 main sections:

- o global -- configuration affecting MPLS behavior which exists independently of the underlying signaling protocol or label switched path configuration.
- o te-global-attributes -- configuration affecting MPLS-TE behavior which exists independently of the underlying signaling protocol or label switched path configuration.
- o signaling protocols -- configuration specific to signaling protocols used to setup and manage label switched paths.
- o label switched paths -- configuration specific to instantiating and managing individual label switched paths.

The top level of the model is shown in the tree view below:

```
+-rw mpls!  
  +-rw global  
  |   ...  
  +-rw te-global-attributes  
  |   ...  
  +-rw signaling-protocols  
  |   ...  
  +-rw lsps  
  |   ...
```

2.1. MPLS global

The global section of the framework provides configuration data for MPLS items which exist independently of an individual label switched path or signaling protocol and are applicable to the MPLS protocol itself. Items such as the depth of the label stack supported, or specific label ranges may be included here.

2.2. TE global attributes

The TE global attributes section of the framework provides configuration control for MPLS-TE items which exist independently of an individual label switched path or signaling protocol. These standalone items are applicable to the entire logical routing device, and establish fundamental configuration such as the threshold for interface bandwidth change that triggers update events into the IGP traffic engineering database (TED). Timers are also specified which determine the length of time an LSP must be present before being considered viable for forwarding use (`te-lsp-install-delay`), and the length of time between LSP teardown and removal of the LSP from the network element's forwarding information base (`te-lsp-cleanup-delay`). Also specified are the name to value mappings of MPLS administrative groups (`mpls-admin-groups`) and shared risk link groups (`mpls-te-srlg`).

```

+--rw te-global-attributes
|   +--rw mpls-te-srlg
|   |   +--rw srlg* [srlg-name]
|   |   |   +--rw srlg-name      leafref
|   |   |   +--rw config
|   |   |   |   +--rw srlg-name?   string
|   |   |   |   +--rw srlg-value?  uint32
|   |   |   |   +--rw srlg-cost?  uint32
|   |   |   +--ro state
|   |   ...
|   |   +--rw members-list* [from-address]
|   |   |   +--rw from-address    leafref
|   |   |   +--rw config
|   |   |   |   +--rw from-address? inet:ip-address
|   |   |   |   +--rw to-address?  inet:ip-address
|   |   |   ...
|   |   +--rw igp-flooding-bandwidth
|   |   |   +--rw config
|   |   |   |   +--rw threshold-type?      enumeration
|   |   |   |   +--rw delta-percentage?   oc-types:percentage
|   |   |   |   +--rw threshold-specification? enumeration
|   |   |   |   +--rw up-thresholds*      oc-types:percentage
|   |   |   |   +--rw down-thresholds*    oc-types:percentage
|   |   |   |   +--rw up-down-thresholds* oc-types:percentage
|   |   |   +--ro state
|   |   ...
|   +--rw mpls-admin-groups
|   |   +--rw admin-group* [admin-group-name]
|   |   |   +--rw admin-group-name    leafref
|   |   |   +--rw config
|   |   |   |   +--rw admin-group-name?  string
|   |   |   |   +--rw admin-group-value? uint32
|   |   |   +--rw state
|   |   ...
|   +--rw te-lsp-timers
|   |   +--rw config
|   |   |   +--rw te-lsp-install-delay?  uint16
|   |   |   +--rw te-lsp-cleanup-delay?  uint16
|   |   |   +--rw te-lsp-reoptimize-timer? uint16
|   |   |   +--ro state
|   |   ...
|   ...

```

2.3. TE interface attributes overview

The TE interface attributes section of the framework provides configuration and state related to traffic engineering such as te-metric or shared risk link group configuration.

```

+--rw te-intf-attributes
|   +--rw interface* [interface-name]
|       +--rw interface-name    leafref
|       +--rw config
|           +--rw interface-name?    ocif:interface-ref
|           +--rw te-metric?         uint32
|           +--rw srlg* [srlg-name]
|               ...
|           +--rw admin-group* [admin-group-name]
|               ...
|           +--rw igp-flooding-bandwidth
|               ...
|       +--ro state
...

```

2.4. Signaling protocol overview

The signaling protocol section of the framework provides configuration elements for configuring three major methods of signaling label switched paths: RSVP-TE, segment routing, and label distribution protocol (LDP). BGP-LU will be included in a future version of this draft by definitions in the BGP model ([I-D.ietf-idr-bgp-model]) and corresponding augmentations to the MPLS model.

```

+--rw signaling-protocols
|   +--rw rsvp-te
|       ...
|   +--rw segment-routing
|       ...
|   +--rw ldp
|       ...

```

Configuration of RSVP-TE is centered around interfaces on the device which participate in the protocol. A key focus is to expose common RSVP-TE configuration parameters which are used to enhance scale and reliability. Items which are applicable globally in the RSVP-TE protocol such as graceful restart, soft preemption and various statistics are grouped into a global section under the protocol. RSVP neighbor and session state are also available in the RSVP section.

```

+--rw rsvp-te
| | +--rw rsvp-sessions
| | | +--rw config
| | | +--ro state
| | |   +--ro rsvp-session* [source-port destination-port
| | |     source-address destination-address]
| | |   ...
| | | +--rw rsvp-neighbors
| | | | +--rw config
| | | | +--ro state
| | | |   +--ro rsvp-neighbor* [neighbor-address]
| | | |   ...
| | | +--rw global
| | | | +--rw graceful-restart
| | | |   ...
| | | | +--rw soft-preemption
| | | |   ...
| | | | +--ro statistics
| | | | | +--ro counters
| | | | | ....
| | | +--rw interface-attributes
| | | | +--rw interface* [interface-name]
| | | | | +--rw interface-name leafref
| | | | | ...
| | | | +--rw rsvp-hellos
| | | | | ...
| | | | +--rw authentication
| | | | | ...
| | | | +--rw subscription
| | | | | ...
| | | | +--rw protection
| | | | | ...
| | | | ...
| | | ...
| | ...
| ...
...

```

Containers for specifying signaling via segment routing and LDP are also present. Specific subelements will be added for those protocols, as well as for BGP labeled unicast, in the next revision.

2.5. LSP overview

This part of the framework contains LSP information. At the high level, LSPs are split into three categories: traffic-engineering-capable (constrained-path), non-traffic-engineered determined by the IGP (unconstrained-path), and hop-by-hop configured (static).

```
+-rw mpls!  
  +-rw lsps  
    +-rw constrained-path  
    |   ...  
  +-rw unconstrained-path  
    |   ...  
  +-rw static-lsps  
    ...
```

The first two categories, constrained-path and unconstrained-path are the ones for which multiple signaling protocols exist, and are organized in protocol-specific and protocol-independent sections. For example, traffic-engineered (constrained path) LSPs may be set up using RSVP-TE or segment routing, and unconstrained LSPs that follow the IGP path may be signaled with LDP or with segment routing. IGP-determined LSPs may also be signaled by RSVP but this usage is not considered in the current version of the model.

A portion of the data model for constrained path traffic-engineered LSPs signaled with RSVP is shown below. It contains configuration for named explicit paths and for tunnels. Tunnel configuration differs for p2p and p2mp LSPs. In either case, some part of the model is signaling-protocol independent. For example for a p2p LSP, attributes such as the path computation method, the constraints for the the path, the bandwidth allocated to it, and even the frequency of reoptimization are signaling-protocol independent, while other data, such as the setup and hold priorities are protocol-specific and are specified in the protocol specific part of the model.

```

+--rw mpls!
  +--rw lsps
    +--rw constrained-path
      | +--rw explicit-path* [name]
        ...
        | +--rw tunnel* [name type]
          | +--rw name          leafref
          | +--rw type          leafref
          | +--rw config
          | | +--rw name?          string
          | | +--rw type?         identityref
          | | +--rw local-id?     union
          | | +--rw description?  string
          | | +--rw admin-status? identityref
          | | +--rw preference?   uint8
          | | +--rw metric?       te-metric-type
          | | +--rw (bandwidth)?
          | | ...
          | | +--rw protection-style-requested? identityref
          | | +--rw te-lsp-reoptimize-timer?   uint16
          | | +--rw (signaling-specific-tunnel-attributes)?
          | | | +--:(RSVP)
          | | | | +--rw source?          inet:ip-address
          | | | | +--rw soft-preemption?  boolean
          | | | +--rw (tunnel-type)?
          | | | | +--:(p2p)
          | | | | | +--rw destination?    inet:ip-address
          | | | | | +--rw primary-paths* [name]
          | | | | | | +--rw name          string
          | | | | | | +--rw preference?   uint8
          | | | | | | +--rw path-computation-method
          | | | | | ...
          | | | | | +--rw admin-groups
          | | | | | ...
          | | | | | +--rw no-cspf?          empty
          | | | | | +--rw (sigaling-specific-path-attributes)?
          | | | | | | +--:(RSVP)
          | | | | | | | +--rw setup-priority?    uint8
          | | | | | | | +--rw hold-priority?     uint8
          | | | | | | | +--rw retry-timer?      uint16
          | | | | | | +--:(SR)
          | | | | | | | +--rw sid-selection-mode?  enumeration
          | | | | | | | +--rw sid-protection-required?  boolean
          | | | | | +--rw secondary-paths* [name]
          | | | | | ...
          | | | +--ro state
          | | | ...

```

Similarly, the partial model for non-traffic-engineered, or IGP-based, LSPs is shown below:

```
+-rw mpls!  
  +-rw lsps  
    +-rw unconstrained-path  
      +-rw path-setup-protocol  
        +-rw ldp!  
          | ...  
          +-rw segment-routing!  
            ...
```

3. Example use cases

3.1. Traffic engineered p2p LSP signaled with RSVP

A possible scenario may be the establishment of a mesh of traffic-engineered LSPs where RSVP signaling is desired, and the LSPs use a local constrained path calculation to determine their path. These LSPs would fall into the category of a constrained-path LSP, and the tunnel type is p2p. Attributes such as metric, bandwidth or the style of protection desired are also defined at this (protocol-independent) level in the model. The path is defined to be locally-computed under the path-computation-method container, specifying the use of CSPF (use-cspf). Additional attributes for the path, such as its RSVP priorities are specified at the path level under the protocol-specific stanza.

```

+--rw mpls!
  +--rw lsps
    +--rw constrained-path
      ...
      | +--rw tunnel* [name type]
      | | +--rw name      leafref
      | | +--rw type      leafref
      | | +--rw config
      | | | +--rw name?          string
      | | | +--rw type?         identityref
      | | | +--rw metric?       te-metric-type
      | | | +--rw (bandwidth)?
      | | | ...
      | | | +--rw protection-style-requested? identityref
      | | | +--rw te-lsp-reoptimize-timer?   uint16
      | | | ...
      | | | +--rw (tunnel-type)?
      | | | +--:(p2p)
      | | | | +--rw destination?          inet:ip-address
      | | | | +--rw primary-paths* [name]
      | | | | | +--rw name                string
      | | | | | +--rw preference?         uint8
      | | | | | +--rw path-computation-method
      | | | | | ...
      | | | | | +--rw admin-groups
      | | | | | ...
      | | | | | +--rw no-cspf?              empty
      | | | | | +--rw (sigaling-specific-path-attributes)?
      | | | | | +--:(RSVP)
      | | | | | | +--rw setup-priority?    uint8
      | | | | | | +--rw hold-priority?    uint8
      | | | | | | +--rw retry-timer?     uint16
      | | | | | +--:(SR)
      | | | | | | +--rw sid-selection-mode? enumeration
      | | | | | | +--rw sid-protection-required? boolean
      | | | | | +--rw secondary-paths* [name]
      | | | | | ...
      | | | +--ro state

```

3.2. Traffic engineered LSP signaled with SR

A possible scenario may be the establishment of disjoint paths in a network where there is no requirement for per-LSP state to be held on midpoint nodes within the network, or RSVP-TE is unsuitable (as described in [I-D.ietf-spring-segment-routing-mpls] and [I-D.shakir-rtgwg-sr-performance-engineered-lsps]). Such LSPs fall in the constrained-path category. Similar to any other traffic engineered LSPs, the path computation method must be specified. Path

attributes, such as the as lsp- placement-constraints (expressed as administrative groups) or metric must be defined. Finally, the path must be specified in a signaling- protocol specific manner appropriate for SR. The same configuration elements from the tree above apply in this case, except that path setup is done by the head-end by building a label stack, rather than signaled.

3.3. IGP-congruent LDP-signaled LSP

A possible scenario may be the establishment of a full mesh of LSPs. When traffic engineering is not an objective, no constraints are placed on the end-to-end path, and the best- effort path can be setup using LDP signaling simply for label distribution. The LSPs follow IGP-computed paths, and fall in the unconstrained-path category in the model. Protocol-specific configuration pertaining to the signaling protocol used, such as the FEC definition and metrics assigned are in the path- setup-protocol portion of the model.

The relevant part of the model for this case is shown below:

```

+--rw mpls!
  +--rw lsps
    +--rw unconstrained-path
      +--rw path-setup-protocol
        +--rw ldp!
          +--rw tunnel
            +--rw tunnel-type?   mplst:tunnel-type
            +--rw ldp-type?      enumeration
            +--rw p2p-lsp
            | +--rw fec-address*  inet:ip-prefix
            +--rw p2mp-lsp
            +--rw mp2mp-lsp

```

A common operational issue encountered when using LDP is traffic blackholing under the following scenario: when an IGP failure occurs, LDP is not aware of it as these are two protocols running independently, resulting in traffic blackholing at the IGP failure point even though LDP is up and running. LDP-IGP synchronization [RFC5443] can be used to cost out the IGP failing point/segment to avoid the blackholing issue. The LDP-IGP synchronization function will be incorporated in a future version of this document.

Note that targeted LDP sessions are not discussed in this use case, and will be incorporated as a separate use case in a future version of this document.

4. Security Considerations

MPLS configuration has a significant impact on network operations, and as such any related protocol or 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 BGP 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.

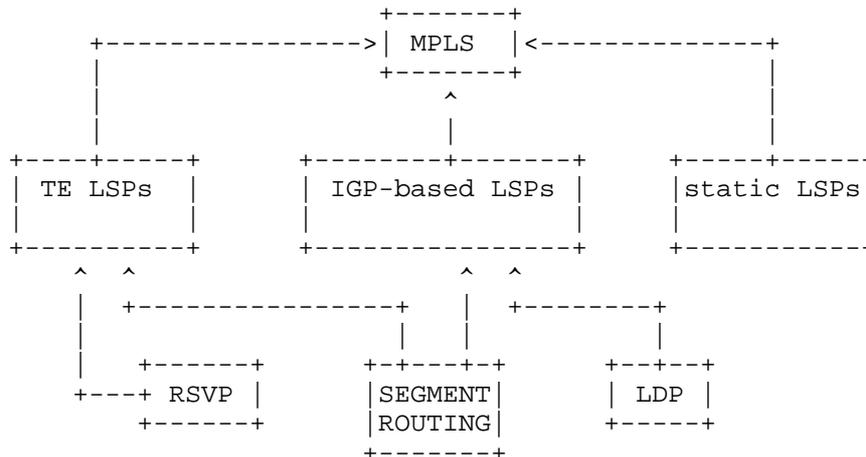
Most of the data elements in the configuration model could be considered sensitive from a security standpoint. Unauthorized access or invalid data could cause major disruption.

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 MPLS YANG modules will be registered in the "YANG Module Names" registry [RFC6020].

6. YANG modules

The modules and submodules comprising the MPLS configuration and operational model are currently organized as depicted below.



The base MPLS module includes submodules describing the three different types of support LSPs, i.e., traffic-engineered (constrained-path), IGP congruent (unconstrained-path), and static. The signaling protocol specific parts of the model are described in separate modules for RSVP, segment routing, and LDP. As mentioned earlier, support for BGP labeled unicast is also planned in a future revision.

A module defining various reusable MPLS types is included, and these modules also make use of the standard Internet types, such as IP addresses, as defined in RFC 6991 [RFC6991].

6.1. MPLS base modules

```
<CODE BEGINS> file openconfig-mpls.yang
module openconfig-mpls {

  yang-version "1";

  // namespace
  namespace "http://openconfig.net/yang/mpls";

  prefix "mpls";

  // import some basic types
  import openconfig-mpls-rsvp { prefix rsvp; }
  import openconfig-mpls-sr { prefix sr; }
  import openconfig-mpls-ldp { prefix ldp; }
  import openconfig-types { prefix oc-types; }
  import openconfig-interfaces { prefix ocif; }

  // include submodules
  include openconfig-mpls-te;
  include openconfig-mpls-igp;
  include openconfig-mpls-static;

  // meta
  organization "OpenConfig working group";

  contact
    "OpenConfig working group
    netopenconfig@googlegroups.com";

  description
    "This module provides data definitions for configuration of
```



```
// identity statements

// grouping statements

grouping mpls-admin-group_config {
  description
    "configuration data for MPLS link admin groups";

  leaf admin-group-name {
    type string;
    description "name for mpls admin-group";
  }

  leaf admin-group-value {
    type uint32;
    description "value for mpls admin-group";
  }
}

grouping mpls-admin-groups-top {

  description "top-level mpls admin-groups config
    and state containers";

  container mpls-admin-groups {
    description
      "Top-level container for admin-groups configuration
      and state";

    list admin-group {
      key admin-group-name;
      description "configuration of value to name mapping
        for mpls affinities/admin-groups";

      leaf admin-group-name {
        type leafref {
          path "../mpls:config/mpls:admin-group-name";
        }
        description
          "name for mpls admin-group";
      }
      container config {
        description "Configurable items for admin-groups";
        uses mpls-admin-group_config;
      }
    }
  }
}
```

```
        container state {
            description "Operational state for admin-groups";
            uses mpls-admin-group_config;
        }
    }
}

grouping mpls-te-igp-flooding-bandwidth_config {
    description
        "Configurable items for igp flooding bandwidth
        threshold configuration.";
    leaf threshold-type {
        type enumeration {
            enum DELTA {
                description "DELTA indicates that the local
                system should flood IGP updates when a
                change in reserved bandwidth >= the specified
                delta occurs on the interface.";
            }
            enum THRESHOLD-CROSSED {
                description "THRESHOLD-CROSSED indicates that
                the local system should trigger an update (and
                hence flood) the reserved bandwidth when the
                reserved bandwidth changes such that it crosses,
                or becomes equal to one of the threshold values.";
            }
        }
    }
    description
        "The type of threshold that should be used to specify the
        values at which bandwidth is flooded. DELTA indicates that
        the local system should flood IGP updates when a change in
        reserved bandwidth >= the specified delta occurs on the
        interface. Where THRESHOLD-CROSSED is specified, the local
        system should trigger an update (and hence flood) the
        reserved bandwidth when the reserved bandwidth changes such
        that it crosses, or becomes equal to one of the threshold
        values";
}

leaf delta-percentage {
    when "../threshold-type = 'DELTA'" {
        description
            "The percentage delta can only be specified when the
            threshold type is specified to be a percentage delta of
            the reserved bandwidth";
    }
    type oc-types:percentage;
}
```

```
description
  "The percentage of the maximum-reservable-bandwidth
  considered as the delta that results in an IGP update
  being flooded";
}

leaf threshold-specification {
  when "../threshold-type = 'THRESHOLD-CROSSED'" {
    description
      "The selection of whether mirrored or separate threshold
      values are to be used requires user specified thresholds to
      be set";
  }
  type enumeration {
    enum MIRRORED-UP-DOWN {
      description
        "MIRRORED-UP-DOWN indicates that a single set of
        threshold values should be used for both increasing
        and decreasing bandwidth when determining whether
        to trigger updated bandwidth values to be flooded
        in the IGP TE extensions.";
    }
    enum SEPARATE-UP-DOWN {
      description
        "SEPARATE-UP-DOWN indicates that a separate
        threshold values should be used for the increasing
        and decreasing bandwidth when determining whether
        to trigger updated bandwidth values to be flooded
        in the IGP TE extensions.";
    }
  }
}
description
  "This value specifies whether a single set of threshold
  values should be used for both increasing and decreasing
  bandwidth when determining whether to trigger updated
  bandwidth values to be flooded in the IGP TE extensions.
  MIRRORED-UP-DOWN indicates that a single value (or set of
  values) should be used for both increasing and decreasing
  values, where SEPARATE-UP-DOWN specifies that the increasing
  and decreasing values will be separately specified";
}

leaf-list up-thresholds {
  when "../threshold-type = 'THRESHOLD-CROSSED'" +
  "and ../threshold-specification = 'SEPARATE-UP-DOWN'" {
    description
      "A list of up-thresholds can only be specified when the
```

```
        bandwidth update is triggered based on crossing a
        threshold and separate up and down thresholds are
        required";
    }
    type oc-types:percentage;
    description
        "The thresholds (expressed as a percentage of the maximum
        reservable bandwidth) at which bandwidth updates are to be
        triggered when the bandwidth is increasing.";
}

leaf-list down-thresholds {
    when "../threshold-type = 'THRESHOLD-CROSSED'" +
        "and ../threshold-specification = 'SEPARATE-UP-DOWN'" {
        description
            "A list of down-thresholds can only be specified when the
            bandwidth update is triggered based on crossing a
            threshold and separate up and down thresholds are
            required";
    }
    type oc-types:percentage;
    description
        "The thresholds (expressed as a percentage of the maximum
        reservable bandwidth) at which bandwidth updates are to be
        triggered when the bandwidth is decreasing.";
}

leaf-list up-down-thresholds {
    when "../threshold-type = 'THRESHOLD-CROSSED'" +
        "and ../threshold-specification = 'MIRRORED-UP-DOWN'" {
        description
            "A list of thresholds corresponding to both increasing
            and decreasing bandwidths can be specified only when an
            update is triggered based on crossing a threshold, and
            the same up and down thresholds are required.";
    }
    type oc-types:percentage;
    description
        "The thresholds (expressed as a percentage of the maximum
        reservable bandwidth of the interface) at which bandwidth
        updates are flooded - used both when the bandwidth is
        increasing and decreasing";
}
}

grouping mpls-te-igp-flooding-bandwidth-if {
    description "Interface-level group for traffic engineering
    database flooding options options";
}
```

```
    container igp-flooding-bandwidth {
      description "Interface bandwidth change percentages
        that trigger update events into the IGP traffic
        engineering database (TED)";
      uses mpls-te-igp-flooding-bandwidth_config;
    }
  }

grouping mpls-te-igp-flooding-bandwidth {
  description "Top level group for traffic engineering
    database flooding options";
  container igp-flooding-bandwidth {
    description "Interface bandwidth change percentages
      that trigger update events into the IGP traffic
      engineering database (TED)";
    container config {
      description "Configuration parameters for TED
        update threshold ";
      uses mpls-te-igp-flooding-bandwidth_config;
    }
    container state {
      config false;
      description "State parameters for TED update threshold ";
      uses mpls-te-igp-flooding-bandwidth_config;
    }
  }
}

grouping te_lsp_delay_config {
  description "Group for the timers goerning the delay
    in installation and cleanup of TE LSPs";

  leaf te-lsp-install-delay {
    type uint16 {
      range 0..3600;
    }
    units seconds;
    description "delay the use of newly installed te lsp for a
      specified amount of time.";
  }

  leaf te-lsp-cleanup-delay {
    type uint16;
    units seconds;
    description "delay the removal of old te lsp for a specified
      amount of time";
  }
}
```

```
    }
  }

  grouping te-interface-attributes-top {
    description
      "Top level grouping for attributes
      for TE interfaces.";

    list interface {
      key interface-name;
      description "List of TE interfaces";

      leaf interface-name {
        type leafref {
          path "../config/interface-name";
          require-instance true;
        }
        description "The interface name";
      }

      container config {
        description
          "Configuration parameters related to TE interfaces:";
        uses te-interface-attributes-config;
      }

      container state {
        config false;
        description "State parameters related to TE interfaces";
        uses te-interface-attributes-config;
      }
    }
  }

  grouping te-interface-attributes-config {
    description "global level definitions for interfaces
    on which TE is run";

    leaf interface-name {
      type ocif:interface-ref;
      description "reference to interface name";
    }

    leaf te-metric {
      type uint32;
      description "TE specific metric for the link";
    }
  }
}
```

```
list srlg {
  key srlg-name;
  description "list of shared risk link groups on the
  interface";
  leaf srlg-name {
    type string;
    description "The SRLG group identifier";
  }
}

list admin-group {
  key admin-group-name;
  description "list of admin groups on the
  interface";
  leaf admin-group-name {
    type string;
    description "The admin group identifier";
  }
}

uses mpls-te-igp-flooding-bandwidth-if;
}

grouping mpls-te-lsp-timers {
  description
  "Grouping for traffic engineering timers";
  container te-lsp-timers {
    description
    "definition for delays associated with setup
    and cleanup of TE LSPs";

    container config {
      description
      "Configuration parameters related
      to timers for TE LSPs";

      uses te_lsp_delay_config;
      uses te-tunnel-reoptimize_config;
    }
    container state {
      config false;
      description "State related to timers for TE LSPs";

      uses te_lsp_delay_config;
      uses te-tunnel-reoptimize_config;
    }
  }
}
```

```
    }  
  
    container mpls {  
        presence "top-level container for MPLS config and operational  
        state";  
  
        description "Anchor point for mpls configuration and operational  
        data";  
  
        container global {  
            // entropy label support, label ranges will be added here.  
            description "general mpls configuration applicable to any  
            type of LSP and signaling protocol - label ranges,  
            entropy label support may be added here";  
        }  
  
        container te-global-attributes {  
            description "traffic-engineering global attributes";  
            uses mpls-te-srlg-top;  
            uses mpls-te-igp-flooding-bandwidth;  
            uses mpls-admin-groups-top;  
            uses mpls-te-lsp-timers;  
        }  
  
        container te-intf-attributes {  
            description "traffic engineering attributes specific  
            for interfaces";  
            uses te-interface-attributes-top;  
        }  
  
        container signaling-protocols {  
            description "top-level signaling protocol configuration";  
  
            uses rsvp:rsvp-global;  
            uses sr:sr-global;  
            uses ldp:ldp-global;  
        }  
  
        container lsps {  
            description "LSP definitions and configuration";  
  
            container constrained-path {  
                description "traffic-engineered LSPs supporting different  
                path computation and signaling methods";  
                uses explicit-paths-top;  
                uses te-tunnels-top;  
            }  
        }  
    }  
}
```

```
    container unconstrained-path {
        description "LSPs that use the IGP-determined path, i.e., non
            traffic-engineered, or non constrained-path";

        uses igp-lsp-common;
        uses igp-lsp-setup;
    }

    container static-lsps {
        description "statically configured LSPs, without dynamic
            signaling";

        uses static-lsp-main;
    }
}

// augment statements

// rpc statements

// notification statements
}

<CODE ENDS>
```

```
    <CODE BEGINS> file openconfig-mpls-types.yang
module openconfig-mpls-types {

    yang-version "1";

    // namespace
    namespace "http://openconfig.net/yang/mpls-types";

    prefix "mplst";

    // meta
    organization "OpenConfig working group";

    contact
        "OpenConfig working group
        netopenconfig@googlegroups.com";

    description
        "General types for MPLS / TE data model";
```

```
revision "2015-10-04" {
  description
    "Work in progress";
  reference "TBD";
}

// extension statements

// feature statements

// identity statements

// using identities rather than enum types to simplify adding new
// signaling protocols as they are introduced and supported
identity path-setup-protocol {
  description "base identity for supported MPLS signaling
  protocols";
}

identity path-setup-rsvp {
  base path-setup-protocol;
  description "RSVP-TE signaling protocol";
}

identity path-setup-sr {
  base path-setup-protocol;
  description "Segment routing";
}

identity path-setup-ldp {
  base path-setup-protocol;
  description "LDP - RFC 5036";
}

identity protection-type {
  description "base identity for protection type";
}

identity unprotected {
  base protection-type;
  description "no protection is desired";
}

identity link-protection-requested {
  base protection-type;
  description "link protection is desired";
}
```

```
identity link-node-protection-requested {
  base protection-type;
  description "node and link protection are both desired";
}

identity lsp-role {
  description
    "Base identity for describing the role of
    label switched path at the current node";
}

identity INGRESS {
  base "lsp-role";
  description
    "Label switched path is an ingress (headend)
    LSP";
}

identity EGRESS {
  base "lsp-role";
  description
    "Label switched path is an egress (tailend)
    LSP";
}

identity TRANSIT {
  base "lsp-role";
  description
    "Label switched path is a transit LSP";
}

identity tunnel-type {
  description
    "Base identity from which specific tunnel types are
    derived.";
}

identity P2P {
  base tunnel-type;
  description
    "TE point-to-point tunnel type.";
}

identity P2MP {
  base tunnel-type;
  description
    "TE point-to-multipoint tunnel type.";
```

```
    }

    identity lsp-oper-status {
        description
            "Base identity for LSP operational status";
    }

    identity DOWN {
        base "lsp-oper-status";
        description
            "LSP is operationally down or out of service";
    }

    identity UP {
        base "lsp-oper-status";
        description
            "LSP is operationally active and available
            for traffic.";
    }

    identity tunnel-admin-status {
        description
            "Base identity for tunnel administrative status";
    }

    identity ADMIN_DOWN {
        base "tunnel-admin-status";
        description
            "LSP is administratively down";
    }

    identity ADMIN_UP {
        base "tunnel-admin-status";
        description
            "LSP is administratively up";
    }

    // typedef statements
    typedef mpls-label {
        type union {
            type uint32 {
                range 16..1048575;
            }
            type enumeration {
                enum IPV4_EXPLICIT_NULL {
                    value 0;
                    description "valid at the bottom of the label stack,
```

```
        indicates that stack must be popped and packet forwarded
        based on IPv4 header";
    }
    enum ROUTER_ALERT {
        value 1;
        description "allowed anywhere in the label stack except
        the bottom, local router delivers packet to the local CPU
        when this label is at the top of the stack";
    }
    enum IPV6_EXPLICIT_NULL {
        value 2;
        description "valid at the bottom of the label stack,
        indicates that stack must be popped and packet forwarded
        based on IPv6 header";
    }
    enum IMPLICIT_NULL {
        value 3;
        description "assigned by local LSR but not carried in
        packets";
    }
    enum ENTROPY_LABEL_INDICATOR {
        value 7;
        description "Entropy label indicator, to allow an LSR
        to distinguish between entropy label and applicaiton
        labels RFC 6790";
    }
}
}
description "type for MPLS label value encoding";
reference "RFC 3032 - MPLS Label Stack Encoding";
}

typedef tunnel-type {
    type enumeration {
        enum P2P {
            description "point-to-point label-switched-path";
        }
        enum P2MP {
            description "point-to-multipoint label-switched-path";
        }
        enum MP2MP {
            description "multipoint-to-multipoint label-switched-path";
        }
    }
}
description "defines the tunnel type for the LSP";
reference
    "RFC 6388 - Label Distribution Protocol Extensions for
    Point-to-Multipoint and Multipoint-to-Multipoint Label Switched
```

```
    Paths
    RFC 4875 - Extensions to Resource Reservation Protocol
    - Traffic Engineering (RSVP-TE) for Point-to-Multipoint TE
    Label Switched Paths (LSPs)";
}

typedef bandwidth-kbps {
    type uint64;
    units kbps;
}

typedef bandwidth-mbps {
    type uint64;
    units mbps;
}

typedef bandwidth-gbps {
    type uint64;
    units gbps;
}

// grouping statements

// data definition statements

// augment statements

// rpc statements

// notification statements
}

    <CODE ENDS>
```

6.2. MPLS LSP submodules

```
    <CODE BEGINS> file openconfig-mpls-te.yang
submodule openconfig-mpls-te {

    yang-version "1";

    belongs-to "openconfig-mpls" {
        prefix "mpls";
    }
}
```

```
// import some basic types
import ietf-inet-types { prefix inet; }
import openconfig-mpls-rsvp { prefix rsvp; }
import openconfig-mpls-sr { prefix sr; }
import openconfig-mpls-types {prefix mplst; }
import openconfig-types { prefix oc-types; }
import ietf-yang-types { prefix yang; }

// meta
organization "OpenConfig working group";

contact
  "OpenConfig working group
  netopenconfig@googlegroups.com";

description
  "Configuration related to constrained-path LSPs and traffic
  engineering. These definitions are not specific to a particular
  signaling protocol or mechanism (see related submodules for
  signaling protocol-specific configuration).";

revision "2015-10-04" {
  description
    "Work in progress";
  reference "TBD";
}

// extension statements

// feature statements

// identity statements

// using identities for path comp method, though enums may also
// be appropriate if we decided these are the primary computation
// mechanisms in future.
identity path-computation-method {
  description
    "base identity for supported path computation
    mechanisms";
}

identity locally-computed {
  base path-computation-method;
  description
    "indicates a constrained-path LSP in which the
    path is computed by the local LER";
}
```

```
}

identity externally-queried {
  base path-computation-method;
  description
    "constrained-path LSP in which the path is
    obtained by querying an external source, such as a PCE server";
}

identity explicitly-defined {
  base path-computation-method;
  description
    "constrained-path LSP in which the path is
    explicitly specified as a collection of strict or/and loose
    hops";
}

// typedef statements

typedef mpls-hop-type {
  type enumeration {
    enum LOOSE {
      description "loose hop in an explicit path";
    }
    enum STRICT {
      description "strict hop in an explicit path";
    }
  }
  description
    "enumerated type for specifying loose or strict
    paths";
}

typedef te-metric-type {
  type union {
    type enumeration {
      enum IGP {
        description
          "set the LSP metric to track the underlying
          IGP metric";
      }
    }
    type uint32;
  }
  description
    "union type for setting the LSP TE metric to a
    static value, or to track the IGP metric";
}
```

```
typedef cspf-tie-breaking {
  type enumeration {
    enum RANDOM {
      description
        "CSPF calculation selects a random path among
         multiple equal-cost paths to the destination";
    }
    enum LEAST_FILL {
      description
        "CSPF calculation selects the path with greatest
         available bandwidth";
    }
    enum MOST_FILL {
      description "CSPF calculation selects the path with the least
         available bandwidth";
    }
  }
  default RANDOM;
  description
    "type to indicate the CSPF selection policy when
     multiple equal cost paths are available";
}

// grouping statements

grouping te-tunnel-reoptimize_config {
  description "Definition for reoptimize timer configuration";
  leaf te-lsp-reoptimize-timer {
    type uint16;
    units seconds;
    description
      "frequency of reoptimization of
       a traffic engineered LSP";
  }
}

grouping path-placement-constraints {
  description
    "Top level grouping for path placement constraints";

  container admin-groups {
    description
      "Include/Exclude constraints for link affinities";
    uses te-lsp-exclude-admin-group_config;
    uses te-lsp-include-any-admin-group_config;
    uses te-lsp-include-all-admin-group_config;
  }
}
```

```
    }

    grouping te-tunnel-bandwidth_config {
      description "Bandwidth configuration for TE LSPs";
      choice bandwidth {
        default explicit;
        description
          "select how bandwidth for the LSP will be
          specified and managed";
        case explicit {
          leaf set-bandwidth {
            type uint32;
            description
              "set bandwidth explicitly, e.g., using
              offline calculation";
          }
        }
        case auto {
          uses te-lsp-auto-bandwidth_config;
        }
      }
    }

    grouping te-lsp-auto-bandwidth_config {
      description "Configuration parameters related to autobandwidth";
      container auto-bandwidth {
        description
          "configure auto-bandwidth operation in
          which devices automatically adjust bandwidth to meet
          requirements";

        leaf enabled {
          type boolean;
          default false;
          description
            "enables mpls auto-bandwidth on the
            lsp";
        }

        leaf min-bw {
          type uint32;
          description
            "set the minimum bandwidth in Mbps for an
            auto-bandwidth LSP";
        }

        leaf max-bw {
          type uint32;
        }
      }
    }
  }
}
```

```
    description
      "set the maximum bandwidth in Mbps for an
      auto-bandwidth LSP";
  }

  leaf adjust-interval {
    type uint32;
    description
      "time in seconds between adjustments to
      LSP bandwidth";
  }

  leaf adjust-threshold {
    type oc-types:percentage;
    description
      "percentage difference between the LSP's
      specified bandwidth and its current bandwidth
      allocation -- if the difference is greater than the
      specified percentage, auto-bandwidth adjustment is
      triggered";
  }

  container overflow {
    description
      "configuration of MPLS overflow bandwidth
      adjustment for the LSP";
    uses te-lsp-overflow_config;
  }

  container underflow {
    description
      "configuration of MPLS underflow bandwidth
      adjustment for the LSP";
    uses te-lsp-underflow_config;
  }
}

grouping te-lsp-overflow_config {
  description
    "configuration for mpls lsp bandwidth
    overflow adjustment";

  leaf enabled {
    type boolean;
    default false;
    description
      "enables mpls lsp bandwidth overflow";
  }
}
```

```
        adjustment on the lsp";
    }

    leaf overflow-threshold {
        type oc-types:percentage;
        description
            "bandwidth percentage change to trigger
            an overflow event";
    }

    leaf trigger-event-count {
        type uint16;
        description
            "number of consecutive overflow sample
            events needed to trigger an overflow adjustment";
    }
}

grouping te-lsp-underflow_config {
    description
        "configuration for mpls lsp bandwidth
        underflow adjustment";

    leaf enabled {
        type boolean;
        default false;
        description
            "enables bandwidth underflow
            adjustment on the lsp";
    }

    leaf underflow-threshold {
        type oc-types:percentage;
        description
            "bandwidth percentage change to trigger
            and underflow event";
    }

    leaf trigger-event-count {
        type uint16;
        description
            "number of consecutive underflow sample
            events needed to trigger an underflow adjustment";
    }
}

grouping te-tunnel-metric_config {
```

```
    description "Configuration parameters related to LSP metric";
    leaf metric {
      type te-metric-type;
      description "LSP metric, either explicit or IGP";
    }
  }

grouping te-lsp-exclude-admin-group_config {
  description
    "Configuration parameters related to admin-groups
    to exclude in path calculation";
  list exclude-groups {

    key exclude-admin-group-name;

    description
      "list of admin-groups to exclude in path calculation";

    leaf exclude-admin-group-name {
      type leafref {
        path "/mpls/te-global-attributes/mpls-admin-groups/" +
          "admin-group/admin-group-name";
      }
      description
        "name of the admin group -- references a defined admin
        group";
    }
  }
}

grouping te-lsp-include-all-admin-group_config {
  description
    "Configuration parameters related to admin-groups
    which all must be included in the path calculation";
  list include-all-groups {

    key all-admin-group-name;
    description
      "list of admin-groups of which all must be included";

    leaf all-admin-group-name {
      type leafref {
        path "/mpls/te-global-attributes/mpls-admin-groups/" +
          "admin-group/admin-group-name";
      }
      description
        "name of the admin group -- references a defined
        admin group";
    }
  }
}
```

```
    }
  }
}

grouping te-lsp-include-any-admin-group_config {
  description
    "Configuration parameters related to admin-groups
    of which one must be included in the path calculation";
  list include-any-groups {

    key any-admin-group-name;
    description
      "list of admin-groups of which one must be included";

    leaf any-admin-group-name {
      type leafref {
        path "/mpls/te-global-attributes/mpls-admin-groups/" +
          "admin-group/admin-group-name";
      }
      description
        "name of the admin group -- references a defined
        admin group";
    }
  }
}

grouping te-tunnel-protection_config {
  description
    "Configuration parameters related to LSP
    protection";
  leaf protection-style-requested {
    type identityref {
      base mplst:protection-type;
    }
    default mplst:unprotected;
    description
      "style of mpls frr protection desired: can be
      link, link-node or unprotected.";
  }
}

grouping te-lsp-comp-explicit {
  description
    "definitions for LSPs in which hops are explicitly
    specified";

  container explicit-path {
    description "LSP with explicit path specification";
  }
}
```

```
    leaf path-name {
      type leafref {
        path "/mpls/lsp/constrained-path/"
          + "explicit-path/config/named-explicit-paths/name";
        require-instance true;
      }
      description "reference to a defined path";
    }
  }
}

grouping te-lsp-comp-queried {
  description "definitons for LSPs computed by querying a remote
  service, e.g., PCE server";

  container queried-path {
    description "LSP with path queried from an external server";

    leaf path-computation-server {
      type inet:ip-address;
      description
        "Address of the external path computation
        server";
    }
  }
}

grouping te-lsp-comp-local {
  description "definitons for locally-computed LSPs";

  container locally-computed {
    description "LSP with path computed by local ingress LSR";

    leaf use-cspf {
      type boolean;
      description "Flag to enable CSPF for locally computed LSPs";
    }
    leaf cspf-tiebreaker {
      type cspf-tie-breaking;
      description
        "Determine the tie-breaking method to choose between
        equally desirable paths during CSFP computation";
    }
  }
}

grouping explicit-route-subobject {
```

```
description
  "The explicit route subobject grouping";
choice type {
  description
    "The explicit route subobject type";
  case ipv4-address {
    description
      "IPv4 address explicit route subobject";
    leaf address {
      type inet:ip-address;
      description "router hop for the LSP path";
    }

    leaf hop-type {
      type mpls-hop-type;
      description "strict or loose hop";
    }
  }

  case label {
    leaf value {
      type uint32;
      description "the label value";
    }
    description
      "The Label ERO subobject";
  }
}

// Explicit paths config somewhat following the IETF model
grouping named-explicit-path_config {
  description
    "Global explicit path configuration
    grouping";
  list named-explicit-paths {
    key "name";
    description
      "A list of explicit paths";
    leaf name {
      type string;
      description
        "A string name that uniquely identifies
        an explicit path";
    }
  }
  list explicit-route-objects {
    key "index";
```

```

    description
      "List of explicit route objects";
    leaf index {
      type uint8 {
        range "0..255";
      }
      description
        "Index of this explicit route object,
         to express the order of hops in path";
    }
    uses explicit-route-subobject;
  }
}

grouping explicit-paths-top {
  description
    "common information for MPLS explicit path definition";
  list explicit-path {
    key name;
    description "Explicit path definition";

    leaf name {
      type leafref {
        path "/mpls/lsp/constrained-path/"
          + "explicit-path/config/named-explicit-paths/name";
        require-instance true;
      }
      description "definition for naming an explicit path";
    }
  }
  container config {
    description "configuration for an explicit path";
    uses named-explicit-path_config;
  }
  container state {
    config false;
    description "operational state for LSP path name";
    uses named-explicit-path_config;
  }
}

grouping mpls-te-srlg_config {
  description
    "Configuration of various attributes associated
     with the SRLG";
}

```

```
leaf srlg-name {
  type string;
  description "SRLG group identifier";
}

leaf srlg-value {
  type uint32;
  description "group ID for the SRLG";
}

leaf srlg-cost {
  type uint32;
  description
    "The cost of the SRLG to the computation
    algorithm";
}
}

grouping mpls-te-srlg-members_config {
  description "Configuration of the membership of the SRLG";

  leaf from-address {
    type inet:ip-address;
    description "IP address of the a-side of the SRLG link";
  }

  leaf to-address {
    type inet:ip-address;
    description "IP address of the z-side of the SRLG link";
  }
}

grouping mpls-te-srlg-top {
  description
    "Top level grouping for MPLS shared
    risk link groups.";
  container mpls-te-srlg {
    description
      "Shared risk link groups attributes";
    list srlg {
      key srlg-name;
      description "List of shared risk link groups";

      leaf srlg-name {
        type leafref {
          path "../config/srlg-name";
          require-instance true;
        }
      }
    }
  }
}
```

```

    description "The SRLG group identifier";
  }

  container config {
    description "Configuration parameters related to the SRLG";
    uses mpls-te-srlg_config;
  }

  container state {
    config false;
    description "State parameters related to the SRLG";
    uses mpls-te-srlg_config;
  }

  list members-list {
    key from-address;
    description
      "List of SRLG members, which are expressed
      as IP address endpoints of links contained in the SRLG";

    leaf from-address {
      type leafref {
        path "../config/from-address";
        require-instance true;
      }
      description "The from address of the link in the SRLG";
    }

    container config {
      description
        "Configuration parameters relating to the
        SRLG members";
      uses mpls-te-srlg-members_config;
    }

    container state {
      config false;
      description
        "State parameters relating to the SRLG
        members";
      uses mpls-te-srlg-members_config;
    }
  }
}

grouping tunnel-path_config {

```

```
description
  "Tunnel path properties grouping";
container path-computation-method {
  description
    "select and configure the way the LSP path is
    computed";

  leaf path-computation {
    type identityref {
      base path-computation-method;
    }
    description "path computation method to use with the LSP";
  }

  uses te-lsp-comp-explicit;
  uses te-lsp-comp-queried;
  uses te-lsp-comp-local;
}

uses path-placement-constraints;

leaf no-cspf {
  type empty;
  description
    "Indicates no CSPF is to be attempted on this
    path.";
}

choice signaling-specific-path-attributes {
  description "Signaling-protocol specific path attributes.";
  case RSVP {
    uses rsvp:rsvp-p2p-path-attributes_config;
  }
  case SR {
    uses sr:sr-path-attributes_config;
  }
}

grouping te-tunnel_config {
  description
    "Configuration parameters relevant to a single
    traffic engineered tunnel.";

  leaf name {
    type string;
    description "The tunnel name";
  }
}
```

```
leaf type {
  type identityref {
    base mplst:tunnel-type;
  }
  description "Tunnel type, p2p or p2mp";
}

leaf local-id {
  type union {
    type uint32;
    type string;
  }
  description
    "locally significant optional identifier for the
    tunnel; may be a numerical or string value";
}

leaf description {
  type string;
  description "optional text description for the tunnel";
}

leaf admin-status {
  type identityref {
    base mplst:tunnel-admin-status;
  }
  default mplst:ADMIN_UP;
  description "TE tunnel administrative state.";
}

leaf preference {
  type uint8 {
    range "1..255";
  }
  description "Specifies a preference for this tunnel.
  A lower number signifies a better preference";
}

uses te-tunnel-metric_config;
uses te-tunnel-bandwidth_config;
uses te-tunnel-protection_config;
uses te-tunnel-reoptimize_config;

choice signaling-specific-tunnel-attributes {
  description "Signaling-protocol specific path attributes.";
  case RSVP {
    uses rsvp:rsvp-p2p-tunnel-attributes_config;
  }
}
```

```
}

choice tunnel-type {
  description
    "Describes tunnel by type type";
  case p2p {
    leaf destination {
      type inet:ip-address;
      description
        "P2P tunnel destination address";
    }
    /* P2P list of path(s) */
    list primary-paths {
      key "name";
      leaf name {
        type string;
        description "Path name";
      }
      description
        "List of primary paths for this
        tunnel.";
      leaf preference {
        type uint8 {
          range "1..255";
        }
        description
          "Specifies a preference for
          this path. The lower the
          number higher the
          preference";
      }
      uses tunnel-path_config;
    }

    list secondary-paths {
      key "name";
      description
        "List of secondary paths for this
        tunnel.";
      leaf name {
        type string;
        description "Path name";
      }
      leaf preference {
        type uint8 {
          range "1..255";
        }
        description

```

```
        "Specifies a preference for
        this path. The lower the
        number higher the
        preference";
    }
    uses tunnel-path_config;
}
}
case p2mp {
    // TODO - complete
}
}
}

grouping te-tunnel_state {
    description
        "Counters and statistical data relevent to a single
        tunnel.";

    leaf oper-status {
        type identityref {
            base mplst:lsp-oper-status;
        }
        description
            "The operational status of the TE tunnel";
    }

    leaf role {
        type identityref {
            base mplst:lsp-role;
        }
        description
            "The lsp role at the current node, whether it is headend,
            transit or tailend.";
    }

    container counters {
        description
            "State data for MPLS label switched paths. This state
            data is specific to a single label switched path.";

        leaf bytes {
            type yang:counter64;
            description
                "Number of bytes that have been forwarded over the
                label switched path.";
        }
    }
}
```

```
leaf packets {
  type yang:counter64;
  description
    "Number of pacets that have been forwarded over the
    label switched path.";
}

leaf path-changes {
  type yang:counter64;
  description
    "Number of path changes for the label switched path";
}

leaf state-changes {
  type yang:counter64;
  description
    "Number of state changes for the label switched path";
}

leaf online-time {
  type yang:date-and-time;
  description
    "Indication of the time the label switched path
    transitioned to an Oper Up or in-service state";
}

leaf current-path-time {
  type yang:date-and-time;
  description
    "Indicates the time the LSP switched onto its
    current path. This is reset upon a LSP path
    change.";
}

leaf next-reoptimization-time {
  type yang:date-and-time;
  description
    "Indicates the next scheduled time the LSP
    will be reoptimized.";
}
}
}

grouping te-tunnels-top {
  description
    "Top level grouping for TE tunnels";

  list tunnel {
```

```
key "name type";
description "List of TE tunnels";

leaf name {
  type leafref {
    path "../config/name";
    require-instance true;
  }
  description "The tunnel name";
}

leaf type {
  type leafref {
    path "../config/type";
    require-instance true;
  }
  description "The tunnel type, p2p or p2mp.";
}

container config {
  description
    "Configuration parameters related to TE tunnels:";
  uses te-tunnel_config;
}

container state {
  config false;
  description "State parameters related to TE interfaces";
  uses te-tunnel_config;
  uses te-tunnel_state;
}
}

// data definition statements

// augment statements

// rpc statements

// notification statements
}

<CODE ENDS>
```

6.3. MPLS signaling protocol modules

```
<CODE BEGINS> file openconfig-mpls-rsvp.yang
module openconfig-mpls-rsvp {

  yang-version "1";

  // namespace
  namespace "http://openconfig.net/yang/rsvp";

  prefix "rsvp";

  // import some basic types
  import ietf-inet-types { prefix inet; }
  import openconfig-mpls-types { prefix mplst; }
  import ietf-yang-types { prefix yang; }
  import openconfig-types { prefix oc-types; }

  // meta
  organization "OpenConfig working group";

  contact
    "OpenConfig working group
     netopenconfig@googlegroups.com";

  description
    "Configuration for RSVP-TE signaling, including global protocol
     parameters and LSP-specific configuration for constrained-path
     LSPs";

  revision "2015-09-18" {
    description
      "Initial revision";
    reference "TBD";
  }

  // extension statements

  // feature statements

  // identity statements

  // typedef statements

  // grouping statements
```

```
grouping mpls-rsvp-soft-preemption_config {
  description "Configuration for MPLS soft preemption";
  leaf enable {
    type boolean;
    default false;
    description "Enables soft preemption on a node.";
  }

  leaf soft-preemption-timeout {
    type uint16 {
      range 0..max;
    }
    // The RFC actually recommends 30 seconds as default.
    default 0;
    description
      "Timeout value for soft preemption to revert
       to hard preemption";
    reference "RFC5712 MPLS-TE soft preemption";
  }
}

grouping mpls-rsvp-soft-preemption {
  description "Top level group for MPLS soft preemption";
  container soft-preemption {
    description
      "Protocol options relating to RSVP
       soft preemption";
    container config {
      description
        "Configuration parameters relating to RSVP
         soft preemption support";
      uses mpls-rsvp-soft-preemption_config;
    }
    container state {
      config false;
      description
        "State parameters relating to RSVP
         soft preemption support";
      uses mpls-rsvp-soft-preemption_config;
    }
  }
}

grouping mpls-rsvp-hellos_config {
  description "RSVP protocol options configuration.";

  leaf hello-interval {
    type uint16 {
```

```
    range 1000..60000;
  }
  units milliseconds;
  default 9000;
  description
    "set the interval in ms between RSVP hello
    messages";
  reference
    "RFC 3209: RSVP-TE: Extensions to RSVP for
    LSP Tunnels.
    RFC 5495: Description of the Resource
    Reservation Protocol - Traffic-Engineered
    (RSVP-TE) Graceful Restart Procedures";
}

leaf refresh-reduction {
  type boolean;
  default true;
  description
    "enables all RSVP refresh reduction message
    bundling, RSVP message ID, reliable message delivery
    and summary refresh";
  reference
    "RFC 2961 RSVP Refresh Overhead Reduction
    Extensions";
}
}

grouping mpls-rsvp-hellos {
  description "Top level grouping for RSVP hellos parameters";
  // TODO: confirm that the described semantics are supported
  // on various implementations. Finer grain configuration
  // will be vendor-specific

  container rsvp-hellos {
    description "Top level container for RSVP hello parameters";
    container config {
      description
        "Configuration parameters relating to RSVP
        hellos";
      uses mpls-rsvp-hellos_config;
    }
    container state {
      config false;
      description "State information associated with RSVP hellos";
      uses mpls-rsvp-hellos_config;
    }
  }
}
```

```
    }

    grouping mpls-rsvp-subscription_config {
      description "RSVP subscription configuration";
      leaf subscription {
        type oc-types:percentage;
        description
          "percentage of the interface bandwidth that
           RSVP can reserve";
      }
    }
  }
  grouping mpls-rsvp-subscription {
    description "Top level group for RSVP subscription options";
    container subscription {
      description
        "Bandwidth percentage reservable by RSVP
         on an interface";
      container config {
        description
          "Configuration parameters relating to RSVP
           subscription options";
        uses mpls-rsvp-subscription_config;
      }
      container state {
        config false;
        description
          "State parameters relating to RSVP
           subscription options";
        uses mpls-rsvp-subscription_config;
      }
    }
  }
}

grouping mpls-rsvp-graceful-restart_config {
  description
    "Configuration parameters relating to RSVP Graceful-Restart";

  leaf enable {
    type boolean;
    default false;
    description "Enables graceful restart on the node.";
  }

  leaf restart-time {
    type uint32;
    description
      "Graceful restart time (seconds).";
    reference
  }
}
```

```
        "RFC 5495: Description of the Resource
        Reservation Protocol - Traffic-Engineered
        (RSVP-TE) Graceful Restart Procedures";
    }
    leaf recovery-time {
        type uint32;
        description
            "RSVP state recovery time";
    }
}

grouping mpls-rsvp-graceful-restart {
    description
        "Top level group for RSVP graceful-restart
        parameters";
    container graceful-restart {
        description "TODO";
        container config {
            description
                "Configuration parameters relating to
                graceful-restart";
            uses mpls-rsvp-graceful-restart_config;
        }
        container state {
            config false;
            description
                "State information associated with
                RSVP graceful-restart";
            uses mpls-rsvp-graceful-restart_config;
        }
    }
}

grouping mpls-rsvp-authentication_config {
    description "RSVP authentication parameters container.";
    leaf enable {
        type boolean;
        default false;
        description "Enables RSVP authentication on the node.";
    }
    leaf authentication-key {
        type string {
            // Juniper supports 1..16 while
            // Cisco has a much bigger range, up to 60.
            length "1..32";
        }
        description
            "authenticate RSVP signaling

```

```
        messages";
    reference
        "RFC 2747: RSVP Cryptographic Authentication";
    }
}

grouping mpls-rsvp-authentication {
    description
        "Top level group for RSVP authentication,
        as per RFC2747";
    container authentication {
        description "TODO";
        container config {
            description
                "Configuration parameters relating
                to authentication";
            uses mpls-rsvp-authentication_config;
        }
        container state {
            config false;
            description
                "State information associated
                with authentication";
            uses mpls-rsvp-authentication_config;
        }
    }
}

grouping mpls-rsvp-protection_config {
    description "RSVP facility (link/node) protection configuration";

    leaf link-protection-style-requested {
        type identityref {
            base mplst:protection-type;
        }
        default mplst:link-node-protection-requested;
        description
            "style of mpls frr protection desired:
            link, link-node, or unprotected";
    }

    leaf bypass-optimize-interval {
        type uint16;
        units seconds;
        description
            "interval between periodic optimization
            of the bypass LSPs";
        // note: this is interface specific on juniper
    }
}
```

```
    // on iox, this is global. need to resolve.
  }
  // to be completed, things like enabling link protection,
  // optimization times, etc.
}

grouping mpls-rsvp-link-protection {
  description "Top level group for RSVP protection";
  container protection {
    description "link-protection (NHOP) related configuration";
    container config {
      description "Configuration for link-protection";
      uses mpls-rsvp-protection_config;
    }
    container state {
      config false;
      description "State for link-protection";
      uses mpls-rsvp-protection_config;
    }
  }
}

grouping mpls-rsvp-error-statistics {
  description "RSVP-TE packet statistics";
  container error {
    description "RSVP-TE error statistics";
    leaf authentication-failure {
      type yang:counter32;
      description
        "Authentication failure count";
    }

    leaf path-error {
      type yang:counter32;
      description
        "Path error to client count";
    }

    leaf resv-error {
      type yang:counter32;
      description
        "Resv error to client count";
    }

    leaf path-timeout {
      type yang:counter32;
      description
        "Path timeout count";
    }
  }
}
```

```
    }

    leaf resv-timeout {
      type yang:counter32;
      description
        "Resv timeout count";
    }

    leaf rate-limit {
      type yang:counter32;
      description
        "Count of packets that were rate limited";
    }

    // TODO - complete the other error statistics
  }
}

grouping mpls-rsvp-protocol-statistics {
  description "RSVP protocol statistics";
  container protocol {
    description "RSVP-TE protocol statistics";
    leaf hello-sent {
      type yang:counter32;
      description
        "Hello sent count";
    }

    leaf hello-rcvd {
      type yang:counter32;
      description
        "Hello received count";
    }

    leaf path-sent {
      type yang:counter32;
      description
        "Path sent count";
    }

    leaf path-rcvd {
      type yang:counter32;
      description
        "Path received count";
    }

    // TODO - To be completed the other packet statistics
  }
}
```

```
    }

    grouping mpls-rsvp-statistics {
        description "Top level grouping for RSVP protocol state";
        uses mpls-rsvp-protocol-state;
    }

    grouping rsvp-global {
        description "Global RSVP protocol configuration";
        container rsvp-te {
            description "RSVP-TE global signaling protocol configuration";

            container rsvp-sessions {
                description "Configuration and state of RSVP sessions";

                container config {
                    description
                        "Configuration of RSVP sessions on the device";
                }

                container state {
                    config false;
                    description
                        "State information relating to RSVP sessions
                        on the device";
                    uses mpls-rsvp-session-state;
                }
            }

            container rsvp-neighbors {
                description
                    "Configuration and state for RSVP neighbors connecting
                    to the device";

                container config {
                    description "Configuration of RSVP neighbor information";
                }

                container state {
                    config false;
                    description
                        "State information relating to RSVP neighbors";
                    uses mpls-rsvp-neighbor-state;
                }
            }

            container global {
                description "Platform wide RSVP configuration and state";
            }
        }
    }
}
```

```
uses mpls-rsvp-graceful-restart;
uses mpls-rsvp-soft-preemption;

container statistics {
  config false;
  description "Platform wide RSVP state, including counters";
  // TODO - reconcile global and per-interface
  // protocol-related statistics

  container counters {
    config false;
    description
      "Platform wide RSVP statistics and counters";
    uses mpls-rsvp-global-protocol-state;
    uses mpls-rsvp-statistics;
  }
}

container interface-attributes {
  // interfaces, bw percentages, hello timers, etc goes here";

  list interface {
    key interface-name;
    description "list of per-interface RSVP configurations";

    // TODO: update to interface ref -- move to separate
    // augmentation.
    leaf interface-name {
      type leafref {
        path "../config/interface-name";
        require-instance true;
      }
      description "references a configured IP interface";
    }
  }

  container config {
    description
      "Configuration of per-interface RSVP parameters";

    leaf interface-name {
      type string;
      description "Name of configured IP interface";
    }
  }

  container state {
```

```
    config false;
    description
      "Per-interface RSVP protocol and state information";
    uses mpls-rsvp-interfaces-state;

    container counters {
      config false;
      description
        "Interface specific RSVP statistics and counters";
      uses mpls-rsvp-protocol-state;
    }
  }
}

uses mpls-rsvp-hellos;
uses mpls-rsvp-authentication;
uses mpls-rsvp-subscription;
uses mpls-rsvp-link-protection;
}
}
}
```

```
grouping rsvp-p2p-tunnel-attributes_config {
  description "properties of RSPP point-to-point paths";

  leaf source {
    type inet:ip-address;
    description
      "tunnel source address";
  }

  leaf soft-preemption {
    type boolean;
    default false;
    description "enables RSVP soft-preemption on this LSP";
  }
}
```

```
grouping rsvp-p2p-path-attributes_config {
  description "properties of RSPP point-to-point paths";
  leaf setup-priority {
    type uint8 {
      range 0..7;
    }
    default 7;
    description
      "preemption priority during LSP setup, lower is
```

```
        higher priority; default 7 indicates that LSP will not
        preempt established LSPs during setup";
reference "RFC 3209 - RSVP-TE: Extensions to RSVP for
LSP Tunnels";
}

leaf hold-priority {
  type uint8 {
    range 0..7;
  }
  default 0;
  description
    "preemption priority once the LSP is established,
    lower is higher priority; default 0 indicates other LSPs
    will not preempt the LSPs once established";
reference "RFC 3209 - RSVP-TE: Extensions to RSVP for
LSP Tunnels";
}

leaf retry-timer {
  type uint16 {
    range 1..600;
  }
  units seconds;
  description
    "sets the time between attempts to establish the
    LSP";
}
}

grouping mpls-rsvp-neighbor-state {
  description "State information for RSVP neighbors";

  list rsvp-neighbor {
    key "neighbor-address";
    description
      "List of RSVP neighbors connecting to the device,
      keyed by neighbor address";

    leaf neighbor-address {
      type inet:ip-address;
      description "Address of RSVP neighbor";
    }

    leaf detected-interface {
      type string;
      description "Interface where RSVP neighbor was detected";
    }
  }
}
```

```
leaf neighbor-status {
  type enumeration {
    enum UP {
      description
        "RSVP hello messages are detected from the neighbor";
    }
    enum DOWN {
      description
        "RSVP neighbor not detected as up, due to a
        communication failure or IGP notification
        the neighbor is unavailable";
    }
  }
  description "Enumeration of possible RSVP neighbor states";
}

leaf neighbor-refresh-reduction {
  type boolean;
  description
    "Support of neighbor for RSVP refresh reduction";
  reference
    "RFC 2961 RSVP Refresh Overhead Reduction
    Extensions";
}
}
}

grouping mpls-rsvp-session-state {
  description "State information for RSVP TE sessions";
  list rsvp-session {
    key "source-port destination-port
    source-address destination-address";
    description "List of RSVP sessions";

    leaf source-address {
      type inet:ip-address;
      description "Origin address of RSVP session";
    }

    leaf destination-address {
      type inet:ip-address;
      description "Destination address of RSVP session";
    }

    leaf source-port {
      type uint16;
      description "RSVP source port";
      reference "RFC 2205";
    }
  }
}
```

```
    }

    leaf destination-port {
      type uint16;
      description "RSVP source port";
      reference "RFC 2205";
    }

    leaf session-state {
      type enumeration {
        enum UP {
          description "RSVP session is up";
        }
        enum DOWN {
          description "RSVP session is down";
        }
      }
      description "Enumeration of RSVP session states";
    }

    leaf session-type {
      type enumeration {
        enum SOURCE {
          description "RSVP session originates on this device";
        }
        enum TRANSIT {
          description "RSVP session transits this device only";
        }
        enum DESTINATION {
          description "RSVP session terminates on this device";
        }
      }
      description "Enumeration of possible RSVP session types";
    }

    leaf tunnel-id {
      type uint16;
      description "Unique identifier of RSVP session";
    }

    leaf label-in {
      type mplst:mpls-label;
      description
        "Incoming MPLS label associated with this RSVP session";
    }

    leaf label-out {
```

```
    type mplst:mpls-label;
    description
      "Outgoing MPLS label associated with this RSVP session";
  }

  leaf-list associated-lsps {
    type leafref {
      path "/mpls/lsp/constrained-path/tunnel/" +
        "config/name";
    }
    description
      "List of label switched paths associated with this RSVP
      session";
  }
} //rsvp-session-state

grouping mplst-rsvp-interfaces-state {
  description "RSVP state information relevant to an interface";

  list bandwidth {
    key priority;
    description
      "Available and reserved bandwidth by priority on
      the interface.";

    leaf priority {
      type uint8 {
        range 0..7;
      }
      description
        "RSVP priority level for LSPs traversing the interface";
    }

    leaf available-bandwidth {
      type mplst:bandwidth-mbps;
      description "Bandwidth currently available";
    }

    leaf reserved-bandwidth {
      type mplst:bandwidth-mbps;
      description "Bandwidth currently reserved";
    }
  }

  leaf highwater-mark {
    type mplst:bandwidth-mbps;
    description "Maximum bandwidth ever reserved";
  }
}
```

```
    }

    leaf active-reservation-count {
      type yang:gauge64;
      description "Number of active RSVP reservations";
    }
  }

grouping mpls-rsvp-global-protocol-state {
  description "RSVP protocol statistics which may not apply
    on an interface, but are significant globally.";

  leaf path-timeouts {
    type yang:counter64;
    description "TODO";
  }

  leaf reservation-timeouts {
    type yang:counter64;
    description "TODO";
  }

  leaf rate-limited-messages {
    type yang:counter64;
    description "RSVP messages dropped due to rate limiting";
  }
}

grouping mpls-rsvp-protocol-state {
  description "RSVP protocol statistics and message counters";
  leaf in-path-messages {
    type yang:counter64;
    description "Number of received RSVP Path messages";
  }

  leaf in-path-error-messages {
    type yang:counter64;
    description "Number of received RSVP Path Error messages";
  }

  leaf in-path-tear-messages {
    type yang:counter64;
    description "Number of received RSVP Path Tear messages";
  }

  leaf in-reservation-messages {
    type yang:counter64;
    description "Number of received RSVP Resv messages";
  }
}
```

```
    }

    leaf in-reservation-error-messages {
      type yang:counter64;
      description "Number of received RSVP Resv Error messages";
    }

    leaf in-reservation-tear-messages {
      type yang:counter64;
      description "Number of received RSVP Resv Tear messages";
    }

    leaf in-rsvp-hello-messages {
      type yang:counter64;
      description "Number of received RSVP hello messages";
    }

    leaf in-rsvp-srefresh-messages {
      type yang:counter64;
      description "Number of received RSVP summary refresh messages";
    }

    leaf in-rsvp-ack-messages {
      type yang:counter64;
      description
        "Number of received RSVP refresh reduction ack
        messages";
    }

    leaf out-path-messages {
      type yang:counter64;
      description "Number of sent RSVP PATH messages";
    }

    leaf out-path-error-messages {
      type yang:counter64;
      description "Number of sent RSVP Path Error messages";
    }

    leaf out-path-tear-messages {
      type yang:counter64;
      description "Number of sent RSVP Path Tear messages";
    }

    leaf out-reservation-messages {
      type yang:counter64;
      description "Number of sent RSVP Resv messages";
    }
  }
```

```
leaf out-reservation-error-messages {
  type yang:counter64;
  description "Number of sent RSVP Resv Error messages";
}

leaf out-reservation-tear-messages {
  type yang:counter64;
  description "Number of sent RSVP Resv Tear messages";
}

leaf out-rsvp-hello-messages {
  type yang:counter64;
  description "Number of sent RSVP hello messages";
}

leaf out-rsvp-srefresh-messages {
  type yang:counter64;
  description "Number of sent RSVP summary refresh messages";
}

leaf out-rsvp-ack-messages {
  type yang:counter64;
  description
    "Number of sent RSVP refresh reduction ack messages";
}
}

// data definition statements

// augment statements

// rpc statements

// notification statements
}

<CODE ENDS>

<CODE BEGINS> file openconfig-mpls-sr.yang
module openconfig-mpls-sr {
  yang-version "1";

  // namespace
```

```
namespace "http://openconfig.net/yang/sr";

prefix "sr";

// import some basic types
import ietf-inet-types { prefix inet; }
import openconfig-mpls-types { prefix mplst; }

// meta
organization "OpenConfig working group";

contact
  "OpenConfig working group
  netopenconfig@googlegroups.com";

description
  "Configuration for MPLS with segment routing-based LSPs,
  including global parameters, and LSP-specific configuration for
  both constrained-path and IGP-congruent LSPs";

revision "2015-10-14" {
  description
    "Work in progress";
  reference "TBD";
}

// extension statements

// feature statements

// identity statements

// typedef statements

grouping srgb_config {

  // Matches the "global" configuration options in
  // draft-litkowski-spring-yang...
  // TODO: request to Stephane for this to be a separate
  // grouping such that it can be included.

  leaf lower-bound {
    type uint32;
    description
      "Lower value in the block.";
  }
  leaf upper-bound {
```

```
        type uint32;
        description
            "Upper value in the block.";
    }
    description
        "List of global blocks to be advertised.";
}

grouping srgb_state {
    description
        "State parameters relating to the SRGB";

    leaf size {
        type uint32;
        description
            "Number of indexes in the SRGB block";
    }
    leaf free {
        type uint32;
        description
            "Number of SRGB indexes that have not yet been allocated";
    }
    leaf used {
        type uint32;
        description
            "Number of SRGB indexes that are currently allocated";
    }
}

// TODO: where do we put LFIB entries?

}

grouping adjacency-sid_config {
    description
        "Configuration related to an Adjacency Segment Identifier
        (SID)";

    // tuned from draft-litkowski-spring-yang
    // TODO: need to send a patch to Stephane

    leaf-list advertise {
        type enumeration {
            enum "PROTECTED" {
                description
                    "Advertise an Adjacency-SID for this interface, which is
                    eligible to be protected using a local protection
                    mechanism on the local LSR. The local protection
                    mechanism selected is dependent upon the configuration
```

```
        of RSVP-TE FRR or LFA elsewhere on the system";
    }
    enum UNPROTECTED {
        description
            "Advertise an Adjacency-SID for this interface, which is
            explicitly excluded from being protected by any local
            protection mechanism";
    }
}
description
    "Specifies the type of adjacency SID which should be
    advertised for the specified entity.";
}

leaf-list groups {
    type uint32;
    description
        "Specifies the groups to which this interface belongs.
        Setting a value in this list results in an additional AdjSID
        being advertised, with the S-bit set to 1. The AdjSID is
        assumed to be protected";
}
}

grouping interface_config {
    description
        "Configuration parameters relating to a Segment Routing
        enabled interface";

    leaf interface {
        type string;
        // TODO: this should be changed to a leafref.
        description
            "Reference to the interface for which segment routing
            configuration is to be applied.";
    }
}

// grouping statements

grouping sr-global {
    description "global segment routing signaling configuration";

    container segment-routing {
        description "SR global signaling config";

        list srgb {
            key "lower-bound upper-bound";
```

```
    uses srgb_config;
    container config {
        description
            "Configuration parameters relating to the Segment Routing
            Global Block (SRGB)";
        uses srgb_config;
    }
    container state {
        config false;
        description
            "State parameters relating to the Segment Routing Global
            Block (SRGB)";
        uses srgb_config;
        uses srgb_state;
    }
    description
        "List of Segment Routing Global Block (SRGB) entries. These
        label blocks are reserved to be allocated as domain-wide
        entries.";
}

list interfaces {
    key "interface";
    uses interface_config;
    container config {
        description
            "Interface configuration parameters for Segment Routing
            relating to the specified interface";
        uses interface_config;
    }
    container state {
        config false;
        description
            "State parameters for Segment Routing features relating
            to the specified interface";
        uses interface_config;
    }
    container adjacency-sid {
        description
            "Configuration for Adjacency SIDs that are related to
            the specified interface";
        container config {
            description
                "Configuration parameters for the Adjacency-SIDs
                that are related to this interface";
            uses adjacency-sid_config;
        }
        container state {
```

```
        config false;
        description
            "State parameters for the Adjacency-SIDs that are
            related to this interface";
        uses adjacency-sid_config;
    }
}
description
    "List of interfaces with associated segment routing
    configuration";
}
}
}

grouping sr-path-attributes_config {
    description
        "Configuration parameters relating to SR-TE LSPs";

    leaf sid-selection-mode {
        type enumeration {
            enum "ADJ-SID-ONLY" {
                description
                    "The SR-TE tunnel should only use adjacency SIDs
                    to build the SID stack to be pushed for the LSP";
            }
            enum "MIXED-MODE" {
                description
                    "The SR-TE tunnel can use a mix of adjacency
                    and prefix SIDs to build the SID stack to be pushed
                    to the LSP";
            }
        }
        default "MIXED-MODE";
        description
            "The restrictions placed on the SIDs to be selected by the
            calculation method for the SR-TE LSP";
    }

    leaf sid-protection-required {
        type boolean;
        default "false";
        description
            "When this value is set to true, only SIDs that are
            protected are to be selected by the calculating method
            for the SR-TE LSP.";
    }
}
}
```

```
grouping sr_fec-address_config {
  description
    "Configuration parameters relating to a FEC that is to be
    advertised by Segment Routing";

  leaf fec-address {
    type inet:ip-prefix;
    description
      "FEC that is to be advertised as part of the Prefix-SID";
  }
}

grouping sr_fec-prefix-sid_config {
  description
    "Configuration parameters relating to the nature of the
    Prefix-SID that is to be advertised for a particular FEC";

  leaf type {
    type enumeration {
      enum "INDEX" {
        description
          "Set when the value of the prefix SID should be specified
          as an off-set from the SRGB's zero-value. When multiple
          SRGBs are specified, the zero-value is the minimum
          of their lower bounds";
      }
      enum "ABSOLUTE" {
        description
          "Set when the value of a prefix SID is specified as the
          absolute value within an SRGB. It is an error to specify
          an absolute value outside of a specified SRGB";
      }
    }
    default "INDEX";
    description
      "Specifies how the value of the Prefix-SID should be
      interpreted - whether as an offset to the SRGB, or as an
      absolute value";
  }

  leaf node-flag {
    type boolean;
    description
      "Specifies that the Prefix-SID is to be treated as a Node-SID
      by setting the N-flag in the advertised Prefix-SID TLV in the
      IGP";
  }
}
```

```
leaf last-hop-behavior {
  type enumeration {
    enum "EXPLICIT-NULL" {
      description
        "Specifies that the explicit null label is to be used
        when the penultimate hop forwards a labelled packet to
        this Prefix-SID";
    }
    enum "UNCHANGED" {
      description
        "Specifies that the Prefix-SID's label value is to be
        left in place when the penultimate hop forwards to this
        Prefix-SID";
    }
    enum "PHP" {
      description
        "Specifies that the penultimate hop should pop the
        Prefix-SID label before forwarding to the eLER";
    }
  }
  description
    "Configuration relating to the LFIB actions for the
    Prefix-SID to be used by the penultimate-hop";
}
}
```

```
grouping igp-tunnel-sr {
  description "defintiions for SR-signaled, IGP-based LSP tunnel
  types";

  container tunnel {
    description "contains configuration stanzas for different LSP
    tunnel types (P2P, P2MP, etc.)";

    leaf tunnel-type {
      type mplst:tunnel-type;
      description "specifies the type of LSP, e.g., P2P or P2MP";
    }

    container p2p-lsp {
      when "tunnel-type = 'P2P'" {
        description "container active when LSP tunnel type is
        point to point";
      }
      description "properties of point-to-point tunnels";

      list fec {
```

```

    key "fec-address";
    uses sr_fec-address_config;

    description
        "List of FECs that are to be originated as SR LSPs";

    container config {
        description
            "Configuration parameters relating to the FEC to be
            advertised by SR";
        uses sr_fec-address_config;
    }
    container state {
        config false;
        description
            "Operational state relating to a FEC advertised by SR";
        uses sr_fec-address_config;
    }
    container prefix-sid {
        description
            "Parameters relating to the Prefix-SID
            used for the originated FEC";

        container config {
            description
                "Configuration parameters relating to the Prefix-SID
                used for the originated FEC";
            uses sr_fec-prefix-sid_config;
        }
        container state {
            config false;
            description
                "Operational state parameters relating to the
                Prefix-SID used for the originated FEC";
            uses sr_fec-prefix-sid_config;
        }
    }
}
}
}
}
}

grouping igp-lsp-sr-setup {
    description "grouping for SR-IGP path setup for IGP-congruent
    LSPs";

    container segment-routing {

```

```
        presence "Presence of this container sets the LSP to use
        SR signaling";

        description "segment routing signaling extensions for
        IGP-congruent LSPs";

        uses igp-tunnel-sr;
    }
}

// data definition statements

// augment statements

// rpc statements

// notification statements
}

        <CODE ENDS>
```

```
        <CODE BEGINS> file openconfig-mpls-ldp.yang
module openconfig-mpls-ldp {

    yang-version "1";

    // namespace
    namespace "http://openconfig.net/yang/ldp";

    prefix "ldp";

    // import some basic types
    import ietf-inet-types { prefix inet; }
    import openconfig-mpls-types { prefix mplst; }

    // meta
    organization "OpenConfig working group";

    contact
        "OpenConfig working group
        netopenconfig@googlegroups.com";

    description
        "Configuration of Label Distribution Protocol global and LSP-
```

```
    specific parameters for IGP-congruent LSPs";

revision "2015-07-04" {
  description
    "Initial revision";
  reference "TBD";
}

// extension statements

// feature statements

// identity statements

// typedef statements

// grouping statements

grouping ldp-global {
  description "global LDP signaling configuration";

  container ldp {
    description "LDP global signaling configuration";

    container timers {
      description "LDP timers";
    }
  }
}

grouping igp-tunnel-ldp {
  description "common defintiions for LDP-signaled LSP tunnel
  types";

  container tunnel {
    description "contains configuration stanzas for different LSP
    tunnel types (P2P, P2MP, etc.)";

    leaf tunnel-type {
      type mplst:tunnel-type;
      description "specifies the type of LSP, e.g., P2P or P2MP";
    }

    leaf ldp-type {
      type enumeration {
        enum BASIC {
          description "basic hop-by-hop LSP";
        }
      }
    }
  }
}
```

```
    }
    enum TARGETED {
        description "tLDP LSP";
    }
}
description "specify basic or targeted LDP LSP";
}

container p2p-lsp {
    when "tunnel-type = 'P2P'" {
        description "container active when LSP tunnel type is
        point to point";
    }

    description "properties of point-to-point tunnels";

    leaf-list fec-address {
        type inet:ip-prefix;
        description "Address prefix for packets sharing the same
        forwarding equivalence class for the IGP-based LSP";
    }
}

container p2mp-lsp {
    when "tunnel-type = 'P2MP'" {
        description "container is active when LSP tunnel type is
        point to multipoint";
    }

    description "properties of point-to-multipoint tunnels";

    // TODO: specify group/source, etc.

}

container mp2mp-lsp {
    when "tunnel-type = 'MP2MP'" {
        description "container is active when LSP tunnel type is
        multipoint to multipoint";
    }

    description "properties of multipoint-to-multipoint tunnels";

    // TODO: specify group/source, etc.

}
}
}
```

```
grouping igp-lsp-ldp-setup {
  description "grouping for LDP setup attributes";

  container ldp {

    presence "Presence of this container sets the LSP to use
    LDP signaling";

    description "LDP signaling setup for IGP-congruent LSPs";

    // include tunnel (p2p, p2mp, ...)

    uses igp-tunnel-ldp;

  }
}

// data definition statements

// augment statements

// rpc statements

// notification statements
}
```

<CODE ENDS>

```
submodule openconfig-mpls-igp {
  yang-version "1";

  belongs-to "openconfig-mpls" {
    prefix "mpls";
  }

  // import some basic types
  import openconfig-mpls-ldp { prefix ldp; }
  import openconfig-mpls-sr { prefix sr; }

  // meta
  organization "OpenConfig working group";
}
```

```
contact
  "OpenConfig working group
  netopenconfig@googlegroups.com";

description
  "Configuration generic configuration parameters for IGP-congruent
  LSPs";

revision "2015-07-04" {
  description
    "Initial revision";
  reference "TBD";
}

// extension statements

// feature statements

// identity statements

// typedef statements

// grouping statements

grouping igp-lsp-common {
  description "common definitions for IGP-congruent LSPs";

  // container path-attributes {
  //   description "general path attribute settings for IGP-based
  //   LSPs";

  //}
}

grouping igp-lsp-setup {
  description "signaling protocol definitions for IGP-based LSPs";

  container path-setup-protocol {
    description "select and configure the signaling method for
    the LSP";

    // uses path-setup-common;
    uses ldp:igp-lsp-ldp-setup;
    uses sr:igp-lsp-sr-setup;
  }
}
```

```
}

// data definition statements

// augment statements

// rpc statements

// notification statements
}

<CODE ENDS>

<CODE BEGINS> file openconfig-mpls-static.yang
submodule openconfig-mpls-static {

  yang-version "1";

  belongs-to "openconfig-mpls" {
    prefix "mpls";
  }

  // import some basic types
  import openconfig-mpls-types {prefix mplst; }
  import ietf-inet-types { prefix inet; }

  // meta
  organization "OpenConfig working group";

  contact
    "OpenConfig working group
    netopenconfig@googlegroups.com";

  description
    "Defines static LSP configuration";

  revision "2015-07-04" {
    description
      "Initial revision";
    reference "TBD";
  }

  // extension statements
```

```
// feature statements
// identity statements
// typedef statements
// grouping statements

grouping static-lsp-common {
  description "common definitions for static LSPs";

  leaf next-hop {
    type inet:ip-address;
    description "next hop IP address for the LSP";
  }

  leaf incoming-label {
    type mplst:mpls-label;
    description "label value on the incoming packet";
  }

  leaf push-label {
    type mplst:mpls-label;
    description "label value to push at the current hop for the
    LSP";
  }
}

grouping static-lsp-main {
  description "grouping for top level list of static LSPs";

  list label-switched-path {
    key name;
    description "list of defined static LSPs";

    leaf name {
      type string;
      description "name to identify the LSP";
    }

    // TODO: separation into ingress, transit, egress may help
    // to figure out what exactly is configured, but need to
    // consider whether implementations can support the
    // separation
    container ingress {
      description "Static LSPs for which the router is an
      ingress node";
    }
  }
}
```

```
    uses static-lsp-common;
  }

  container transit {
    description "static LSPs for which the router is a
      transit node";

    uses static-lsp-common;
  }

  container egress {
    description "static LSPs for which the router is a
      egress node";

    uses static-lsp-common;
  }
}

// data definition statements

// augment statements

// rpc statements

// notification statements

}

<CODE ENDS>
```

7. Contributing Authors

The following people contributed significantly to this document and are listed below:

Ina Minei
Google
1600 Amphitheatre Parkway
Mountain View, CA 94043
US
Email: inaminei@google.com

Anees Shaikh
Google
1600 Amphitheatre Parkway
Mountain View, CA 94043
US

Email: aashaikh@google.com

Phil Bedard
Cox Communications
Atlanta, GA 30319
US
Email: phil.bedard@cox.com

8. Acknowledgements

The authors are grateful for valuable contributions to this document and the associated models from: Ebben Aires, Deepak Bansal, Nabil Bitar, Feihong Chen, Mazen Khaddam.

9. References

[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.

[I-D.ietf-spring-segment-routing-mpls]

Filsfils, C., Previdi, S., Bashandy, A., Decraene, B., Litkowski, S., Horneffer, M., rjs@rob.sh, r., Tantsura, J., and E. Crabbe, "Segment Routing with MPLS data plane", draft-ietf-spring-segment-routing-mpls-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.

[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.

- [I-D.shakir-rtgwg-sr-performance-engineered-lsps]
Shakir, R., Vernals, D., and A. Capello, "Performance Engineered LSPs using the Segment Routing Data-Plane", draft-shakir-rtgwg-sr-performance-engineered-lsps-00 (work in progress), July 2013.
- [RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <<http://www.rfc-editor.org/info/rfc3688>>.
- [RFC5443] Jork, M., Atlas, A., and L. Fang, "LDP IGP Synchronization", RFC 5443, DOI 10.17487/RFC5443, March 2009, <<http://www.rfc-editor.org/info/rfc5443>>.
- [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>>.
- [RFC6991] Schoenwaelder, J., Ed., "Common YANG Data Types", RFC 6991, DOI 10.17487/RFC6991, July 2013, <<http://www.rfc-editor.org/info/rfc6991>>.

Authors' Addresses

Joshua George
Google
1600 Amphitheatre Pkwy
Mountain View, CA 94043
US

Email: jgeorge@google.com

Luyuan Fang
Microsoft
15590 NE 31st St
Redmond, WA 98052
US

Email: lufang@microsoft.com

Eric Osborne
Level 3

Email: eric.osborne@level3.com

Rob Shakir
Jive Communications, Inc.
1275 West 1600 North, Suite 100
Orem, UT 84057

Email: rjs@rob.sh

Routing Area Working Group
Internet-Draft
Intended status: Informational
Expires: January 9, 2017

P. Sarkar, Ed.
Individual
S. Hegde
C. Bowers
Juniper Networks, Inc.
U. Chunduri, Ed.
Ericsson Inc.
J. Tantsura
Individual
B. Decraene
Orange
H. Gredler
Unaffiliated
July 8, 2016

LFA selection for Multi-Homed Prefixes
draft-psarkar-rtwg-multihomed-prefix-lfa-04

Abstract

This document shares experience gained from implementing algorithms to determine Loop-Free Alternates for multi-homed prefixes. In particular, this document provides explicit inequalities that can be used to evaluate neighbors as a potential alternates for multi-homed prefixes. It also provides detailed criteria for evaluating potential alternates for external prefixes advertised by OSPF ASBRs.

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 [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 January 9, 2017.

Copyright Notice

Copyright (c) 2016 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction	3
1.1. Acronyms	3
2. LFA inequalities for MHPs	4
3. LFA selection for the multi-homed prefixes	4
3.1. Improved coverage with simplified approach to MHPs	6
3.2. IS-IS ATT Bit considerations	7
4. LFA selection for the multi-homed external prefixes	8
4.1. IS-IS	8
4.2. OSPF	8
4.2.1. Rules to select alternate ASBR	8
4.2.2. Multiple ASBRs belonging different area	9
4.2.3. Type 1 and Type 2 costs	10
4.2.4. RFC1583compatibility is set to enabled	10
4.2.5. Type 7 routes	10
4.2.6. Inequalities to be applied for alternate ASBR selection	10
4.2.6.1. Forwarding address set to non zero value	10
4.2.6.2. ASBRs advertising type1 and type2 cost	11
5. LFA Extended Procedures	12
5.1. Links with IGP MAX_METRIC	12
5.2. Multi Topology Considerations	13
6. Acknowledgements	14
7. IANA Considerations	14
8. Security Considerations	14
9. References	14
9.1. Normative References	14
9.2. Informative References	15
Authors' Addresses	15

1. Introduction

The use of Loop-Free Alternates (LFA) for IP Fast Reroute is specified in [RFC5286]. Section 6.1 of [RFC5286] describes a method to determine loop-free alternates for a multi-homed prefixes (MHPs). This document describes a procedure using explicit inequalities that can be used by a computing router to evaluate a neighbor as a potential alternate for a multi-homed prefix. The results obtained are equivalent to those obtained using the method described in Section 6.1 of [RFC5286]. However, some may find this formulation useful.

Section 6.3 of [RFC5286] discusses complications associated with computing LFAs for multi-homed prefixes in OSPF. This document provides detailed criteria for evaluating potential alternates for external prefixes advertised by OSPF ASBRs, as well as explicit inequalities.

This document also provide clarifications, additional considerations to [RFC5286], to address a few coverage and operational observations. These observations are in the area of handling IS-IS attach (ATT) bit in Level-1 (L1) area, links provisioned with MAX_METRIC for traffic engineering (TE) purposes and in the area of Multi Topology (MT) IGP deployments. All these are elaborated in detail in Section 3.2 and Section 5.

1.1. Acronyms

AF	-	Address Family
ATT	-	IS-IS Attach Bit
ECMP	-	Equal Cost Multi Path
IGP	-	Interior Gateway Protocol
IS-IS	-	Intermediate System to Intermediate System
OSPF	-	Open Shortest Path First
MHP	-	Multi-homed Prefix
MT	-	Multi Topology
SPF	-	Shortest Path First PDU

2. LFA inequalities for MHPs

This document proposes the following set of LFA inequalities for selecting the most appropriate LFAs for multi-homed prefixes (MHPs). They can be derived from the inequalities in [RFC5286] combined with the observation that $D_{opt}(N,P) = \text{Min} (D_{opt}(N,PO_i) + \text{cost}(PO_i,P))$ over all PO_i

Link-Protection:

$$D_{opt}(N,PO_i) + \text{cost}(PO_i,P) < D_{opt}(N,S) + D_{opt}(S,PO_{best}) + \text{cost}(PO_{best},P)$$

Link-Protection + Downstream-paths-only:

$$D_{opt}(N,PO_i) + \text{cost}(PO_i,P) < D_{opt}(S,PO_{best}) + \text{cost}(PO_{best},P)$$

Node-Protection:

$$D_{opt}(N,PO_i) + \text{cost}(PO_i,P) < D_{opt}(N,E) + D_{opt}(E,PO_{best}) + \text{cost}(PO_{best},P)$$

Where,

- S - The computing router
- N - The alternate router being evaluated
- E - The primary next-hop on shortest path from S to prefix P.
- PO_i - The specific prefix-originating router being evaluated.
- PO_{best} - The prefix-originating router on the shortest path from the computing router S to prefix P.
- $\text{Cost}(X,P)$ - Cost of reaching the prefix P from prefix originating node X.
- $D_{opt}(X,Y)$ - Distance on the shortest path from node X to node Y.

Figure 1: LFA inequalities for MHPs

3. LFA selection for the multi-homed prefixes

To compute a valid LFA for a given multi-homed prefix P, through an alternate neighbor N a computing router S MUST follow one of the appropriate procedures below.

Link-Protection :

=====

1. If alternate neighbor N is also prefix-originator of P,
 - 1.a. Select N as a LFA for prefix P (irrespective of the metric advertised by N for the prefix P).
2. Else, evaluate the link-protecting LFA inequality for P with the N as the alternate neighbor.
 - 2.a. If LFA inequality condition is met, select N as a LFA for prefix P.
 - 2.b. Else, N is not a LFA for prefix P.

Link-Protection + Downstream-paths-only :

=====

1. Evaluate the link-protecting + downstream-only LFA inequality for P with the N as the alternate neighbor.
 - 1.a. If LFA inequality condition is met, select N as a LFA for prefix P.
 - 1.b. Else, N is not a LFA for prefix P.

Node-Protection :

=====

1. If alternate neighbor N is also prefix-originator of P,
 - 1.a. Select N as a LFA for prefix P (irrespective of the metric advertised by N for the prefix P).
2. Else, evaluate the appropriate node-protecting LFA inequality for P with the N as the alternate neighbor.
 - 2.a. If LFA inequality condition is met, select N as a LFA for prefix P.
 - 2.b. Else, N is not a LFA for prefix P.

Figure 2: Rules for selecting LFA for MHPs

In case an alternate neighbor N is also one of the prefix-originators of prefix P, N MAY be selected as a valid LFA for P.

However if N is not a prefix-originator of P, the computing router SHOULD evaluate one of the corresponding LFA inequalities, as mentioned in Figure 1, once for each remote node that originated the prefix. In case the inequality is satisfied by the neighbor N router S MUST choose neighbor N, as one of the valid LFAs for the prefix P.

When computing a downstream-only LFA, in addition to being a prefix-originator of P, router N MUST also satisfy the downstream-only LFA inequality specified in Figure 1.

For more specific rules please refer to the later sections of this document.

3.1. Improved coverage with simplified approach to MHPs

LFA base specification [RFC5286] Section 6.1 recommends that a router compute the alternate next-hop for an IGP multi-homed prefix by considering alternate paths via all routers that have announced that prefix and the same has been elaborated with appropriate inequalities in the above section. However, [RFC5286] Section 6.1 also allows for the router to simplify the multi-homed prefix calculation by assuming that the MHP is solely attached to the router that was its pre-failure optimal point of attachment, at the expense of potentially lower coverage. If an implementation chooses to simplify the multi-homed prefix calculation by assuming that the MHP is solely attached to the router that was its pre-failure optimal point of attachment, the procedure described in this memo can potentially improve coverage for equal cost multi path (ECMP) MHPs without incurring extra computational cost.

While the approach as specified in [RFC5286] Section 6.1 last paragraph, is to simplify the MHP as solely attached to the router that was its pre-failure optimal point of attachment; though it is a scalable approach and simplifies computation, [RFC5286] notes this may result in little less coverage.

This memo improves the above approach to provide loop-free alternatives without any additional cost for equal cost multi path MHPs as described through the below example network. The approach specified here MAY also be applicable for handling default routes as explained in Section 3.2.

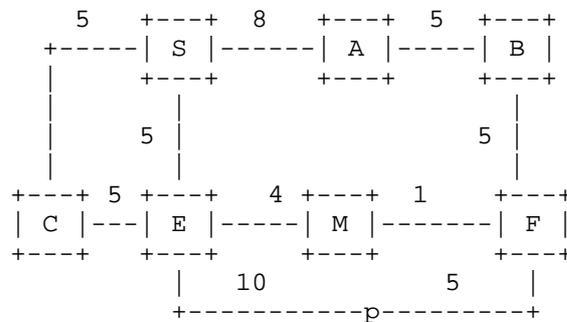


Figure 3: MHP with same ECMP Next-hop

In the above network a prefix p, is advertised from both Node E and Node F. With simplified approach taken as specified in [RFC5286] Section 6.1, prefix p will get only link protection LFA through the neighbor C while a node protection path is available through neighbor

A. In this scenario, E and F both are pre-failure optimal points of attachment and share the same primary next-hop. Hence, an implementation MAY compare the kind of protection A provides to F (link-and-node protection) with the kind of protection C provides to E (link protection) and inherit the better alternative to prefix p and here it is A.

However, in the below network prefix p has an ECMP through both node E and node F with cost 20. Though it has 2 pre-failure optimal points of attachment, the primary next-hop to each pre-failure optimal point of attachment is different. In this case, prefix p shall inherit corresponding LFA to each primary next-hop calculated for the router advertising the same respectively (node E's and node F's LFA).

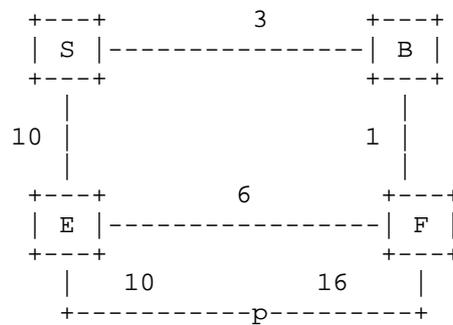


Figure 4: MHP with different ECMP Next-hops

In summary, if there are multiple pre-failure points of attachment for a MHP and primary next-hop of a MHP is same as that of the primary next-hop of the router that was pre-failure optimal point of attachment, an implementation MAY provide the better protection to MHP without incurring any additional computation cost.

3.2. IS-IS ATT Bit considerations

Per [RFC1195] a default route needs to be added in Level1 (L1) router to the closest reachable Level1/Level2 (L1/L2) router in the network advertising ATT (attach) bit in its LSP-0 fragment. All L1 routers in the area would do this during the decision process with the next-hop of the default route set to the adjacent router through which the closest L1/L2 router is reachable. The base LFA specification [RFC5286] does not specify any procedure for computing LFA for a default route in IS-IS L1 area. Potentially one MAY consider a default route is being advertised from the border L1/L2 router where ATT bit is set and can do LFA computation for the default route.

But, when multiple ECMP L1/L2 routers are reachable in an L1 area corresponding best LFAs SHOULD be given for each primary next-hop associated with default route. Considerations as specified in Section 3 and Section 3.1 are applicable for default routes, if the default route is considered as ECMP MHP.

4. LFA selection for the multi-homed external prefixes

Redistribution of external routes into IGP is required in case of two different networks getting merged into one or during protocol migrations. External routes could be distributed into an IGP domain via multiple nodes to avoid a single point of failure.

During LFA calculation, alternate LFA next-hops to reach the best ASBR could be used as LFA for the routes redistributed via that ASBR. When there is no LFA available to the best ASBR, it may be desirable to consider the other ASBRs (referred to as alternate ASBR hereafter) redistributing the external routes for LFA selection as defined in [RFC5286] and leverage the advantage of having multiple redistributing nodes in the network.

4.1. IS-IS

LFA evaluation for multi-homed external prefixes in IS-IS is similar to the multi-homed internal prefixes. Inequalities described in sec 2 would also apply to multi-homed external prefixes as well.

4.2. OSPF

Loop free Alternates [RFC 5286] describes mechanisms to apply inequalities to find the loop free alternate neighbor. For the selection of alternate ASBR for LFA consideration, additional rules have to be applied in selecting the alternate ASBR due to the external route calculation rules imposed by [RFC 2328].

This document also defines the inequalities defined in RFC [5286] specifically for the alternate loop-free ASBR evaluation.

4.2.1. Rules to select alternate ASBR

The process to select an alternate ASBR is best explained using the rules below. The below process is applied when primary ASBR for the concerned prefix is chosen and there is an alternate ASBR originating same prefix.

1. If RFC1583Compatibility is disabled
 - 1a. if primary ASBR and alternate ASBR are intra area non-backbone path go to step 2.
 - 1b. If primary ASBR and alternate ASBR belong to intra-area backbone and/or inter-area path go to step 2.
 - 1c. for other paths, skip the alternate ASBR and consider next ASBR.
2. If cost type (type1/type2) advertised by alternate ASBR same as primary
 - 2a. If not same skip alternate ASBR and consider next ASBR.
3. If cost type is type1
 - 3a. If cost is same, program ECMP
 - 3b. else go to step 5.
4. If cost type is type 2
 - 4a. If cost is different, skip alternate ASBR and consider next ASBR
 - 4b. If type2 cost is same, compare type 1 cost.
 - 4c. If type1 cost is also same program ECMP.
 - 4d. If type 1 cost is different go to step 5.
5. If route type (type 5/type 7)
 - 5a. If route type is same, check route p-bit, forwarding address field for routes from both ASBRs match. If not skip alternate ASBR and consider next ASBR.
 - 5b. If route type is not same, skip ASBR and consider next ASBR.
6. Apply inequality on the alternate ASBR.

Figure 5: Rules for selecting alternate ASBR in OSPF

4.2.2. Multiple ASBRs belonging different area

When "RFC1583compatibility" is set to disabled, OSPF[RFC2328] defines certain rules of preference to choose the ASBRs. While selecting alternate ASBR for loop evaluation for LFA, these rules should be applied and ensured that the alternate neighbor does not loop the traffic back.

When there are multiple ASBRs belonging to different area advertising the same prefix, pruning rules as defined in RFC 2328 section 16.4.1

are applied. The alternate ASBRs pruned using above rules are not considered for LFA evaluation.

4.2.3. Type 1 and Type 2 costs

If there are multiple ASBRs not pruned via rules defined in 3.2.2, the cost type advertised by the ASBRs is compared. ASBRs advertising Type1 costs are preferred and the type2 costs are pruned. If two ASBRs advertise same type2 cost, the alternate ASBRs are considered along with their type1 cost for evaluation. If the two ASBRs with same type2 as well as type1 cost, ECMP FRR is programmed. If there are two ASBRs with different type2 cost, the higher cost ASBR is pruned. The inequalities for evaluating alternate ASBR for type 1 and type 2 costs are same, as the alternate ASBRs with different type2 costs are pruned and the evaluation is based on equal type 2 cost ASBRs.

4.2.4. RFC1583compatibility is set to enabled

When RFC1583Compatibility is set to enabled, multiple ASBRs belonging to different area advertising same prefix are chosen based on cost and hence are valid alternate ASBRs for the LFA evaluation.

4.2.5. Type 7 routes

Type 5 routes always get preference over Type 7 and the alternate ASBRs chosen for LFA calculation should belong to same type. Among Type 7 routes, routes with p-bit and forwarding address set have higher preference than routes without these attributes. Alternate ASBRs selected for LFA comparison should have same p-bit and forwarding address attributes.

4.2.6. Inequalities to be applied for alternate ASBR selection

The alternate ASBRs selected using above mechanism described in 3.2.1, are evaluated for Loop free criteria using below inequalities.

4.2.6.1. Forwarding address set to non zero value

Link-Protection:

$$F_{\text{opt}}(N, PO_i) + \text{cost}(PO_i, P) < D_{\text{opt}}(N, S) + F_{\text{opt}}(S, PO_{\text{best}}) + \text{cost}(PO_{\text{best}}, P)$$

Link-Protection + Downstream-paths-only:

$$F_{\text{opt}}(N, PO_i) + \text{cost}(PO_i, P) < F_{\text{opt}}(S, PO_{\text{best}}) + \text{cost}(PO_{\text{best}}, P)$$

Node-Protection:

$$F_{\text{opt}}(N, PO_i) + \text{cost}(PO_i, P) < D_{\text{opt}}(N, E) + F_{\text{opt}}(E, PO_{\text{best}}) + \text{cost}(PO_{\text{best}}, P)$$

Where,

- S - The computing router
- N - The alternate router being evaluated
- E - The primary next-hop on shortest path from S to prefix P.
- PO_i - The specific prefix-originating router being evaluated.
- PO_{best} - The prefix-originating router on the shortest path from the computing router S to prefix P.
- cost(X,Y) - External cost for Y as advertised by X
- F_{opt}(X,Y) - Distance on the shortest path from node X to Forwarding address specified by ASBR Y.
- D_{opt}(X,Y) - Distance on the shortest path from node X to node Y.

Figure 6: LFA inequality definition when forwarding address in non-zero

4.2.6.2. ASBRs advertising type1 and type2 cost

Link-Protection:

$$D_{\text{opt}}(N, PO_i) + \text{cost}(PO_i, P) < D_{\text{opt}}(N, S) + D_{\text{opt}}(S, PO_{\text{best}}) + \text{cost}(PO_{\text{best}}, P)$$

Link-Protection + Downstream-paths-only:

$$D_{\text{opt}}(N, PO_i) + \text{cost}(PO_i, P) < D_{\text{opt}}(S, PO_{\text{best}}) + \text{cost}(PO_{\text{best}}, P)$$

Node-Protection:

$$D_{\text{opt}}(N, PO_i) + \text{cost}(PO_i, P) < D_{\text{opt}}(N, E) + D_{\text{opt}}(E, PO_{\text{best}}) + \text{cost}(PO_{\text{best}}, P)$$

Where,

- S - The computing router
- N - The alternate router being evaluated
- E - The primary next-hop on shortest path from S to prefix P.
- PO_i - The specific prefix-originating router being evaluated.
- PO_{best} - The prefix-originating router on the shortest path from the computing router S to prefix P.
- cost(X,Y) - External cost for Y as advertised by X.
- D_{opt}(X,Y) - Distance on the shortest path from node X to node Y.

Figure 7: LFA inequality definition for type1 and type 2 cost

5. LFA Extended Procedures

This section explains the additional considerations in various aspects as listed below to the base LFA specification [RFC5286].

5.1. Links with IGP MAX_METRIC

Section 3.5 and 3.6 of [RFC5286] describes procedures for excluding nodes and links from use in alternate paths based on the maximum link metric (as defined in for IS-IS in [RFC5305] or as defined in [RFC3137] for OSPF). If these procedures are strictly followed, there are situations, as described below, where the only potential alternate available which satisfies the basic loop-free condition will not be considered as alternative.

determined by the presence of IGP adjacency from Source to the neighboring node on that MT-ID apart from the administrative restrictions and other checks laid out in [RFC5286]. The same is also applicable for OSPF [RFC4915] [MT-OSPF] or different AFs in multi instance OSPFv3 [RFC5838].

However for MT IS-IS, if a default topology is used with MT-ID 0 [RFC5286] and both IPv4 [RFC5305] and IPv6 routes/AFs [RFC5308] are present, then the condition of network congruency is applicable for LFA computation as well. Network congruency here refers to, having same address families provisioned on all the links and all the nodes of the network with MT-ID 0. Here with single decision process both IPv4 and IPv6 next-hops are computed for all the prefixes in the network and similarly with one LFA computation from all eligible neighbors per [RFC5286], all potential alternatives can be computed.

6. Acknowledgements

Thanks to Alia Atlas and Salih K A for their useful feedback and inputs.

7. IANA Considerations

N/A. - No protocol changes are proposed in this document.

8. Security Considerations

This document does not introduce any change in any of the protocol specifications and also this does not introduce any new security issues other than as noted in the LFA base specification [RFC5286].

9. References

9.1. Normative References

- [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>>.
- [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>>.

9.2. Informative References

- [I-D.ietf-rtgwg-lfa-manageability]
Litkowski, S., Decraene, B., Filsfils, C., Raza, K., and M. Horneffer, "Operational management of Loop Free Alternates", draft-ietf-rtgwg-lfa-manageability-11 (work in progress), June 2015.
- [RFC3137] Retana, A., Nguyen, L., White, R., Zinin, A., and D. McPherson, "OSPF Stub Router Advertisement", RFC 3137, DOI 10.17487/RFC3137, June 2001, <<http://www.rfc-editor.org/info/rfc3137>>.
- [RFC4915] Psenak, P., Mirtorabi, S., Roy, A., Nguyen, L., and P. Pillay-Esnault, "Multi-Topology (MT) Routing in OSPF", RFC 4915, DOI 10.17487/RFC4915, June 2007, <<http://www.rfc-editor.org/info/rfc4915>>.
- [RFC5120] Przygienda, T., Shen, N., and N. Sheth, "M-ISIS: Multi Topology (MT) Routing in Intermediate System to Intermediate Systems (IS-ISs)", RFC 5120, DOI 10.17487/RFC5120, February 2008, <<http://www.rfc-editor.org/info/rfc5120>>.
- [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>>.
- [RFC5305] Li, T. and H. Smit, "IS-IS Extensions for Traffic Engineering", RFC 5305, DOI 10.17487/RFC5305, October 2008, <<http://www.rfc-editor.org/info/rfc5305>>.
- [RFC5308] Hopps, C., "Routing IPv6 with IS-IS", RFC 5308, DOI 10.17487/RFC5308, October 2008, <<http://www.rfc-editor.org/info/rfc5308>>.
- [RFC5838] Lindem, A., Ed., Mirtorabi, S., Roy, A., Barnes, M., and R. Aggarwal, "Support of Address Families in OSPFv3", RFC 5838, DOI 10.17487/RFC5838, April 2010, <<http://www.rfc-editor.org/info/rfc5838>>.

Authors' Addresses

Pushpasis Sarkar (editor)
Individual

Email: pushpasis.ietf@gmail.com

Shraddha Hegde
Juniper Networks, Inc.
Electra, Exora Business Park
Bangalore, KA 560103
India

Email: shraddha@juniper.net

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

Email: cbowers@juniper.net

Uma Chunduri (editor)
Ericsson Inc.
300 Holger Way,
San Jose, California 95134
USA

Phone: 408 750-5678
Email: uma.chunduri@ericsson.com

Jeff Tantsura
Individual

Email: jefftant.ietf@gmail.com

Bruno Decraene
Orange

Email: bruno.decraene@orange.com

Hannes Gredler
Unaffiliated

Email: hannes@gredler.at

RTGWG
Internet-Draft
Intended status: Informational
Expires: November 22, 2015

E. Nordmark (ed)
Arista Networks
A. Tian
Ericsson Inc.
J. Gross
VMware
J. Hudson
Brocade Communications Systems,
Inc.
L. Kreeger
Cisco Systems, Inc.
P. Garg
Microsoft
P. Thaler
Broadcom Corporation
T. Herbert
Google
May 21, 2015

Encapsulation Considerations
draft-rtg-dt-encap-02

Abstract

The IETF Routing Area director has chartered a design team to look at common issues for the different data plane encapsulations being discussed in the NVO3 and SFC working groups and also in the BIER BoF, and also to look at the relationship between such encapsulations in the case that they might be used at the same time. The purpose of this design team is to discover, discuss and document considerations across the different encapsulations in the different WGs/BoFs so that we can reduce the number of wheels that need to be reinvented in the future.

Status of this Memo

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

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

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

material or to cite them other than as "work in progress."

This Internet-Draft will expire on November 22, 2015.

Copyright Notice

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

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

Table of Contents

1. Design Team Charter	4
2. Overview	4
3. Common Issues	6
4. Scope	6
5. Assumptions	7
6. Terminology	8
7. Entropy	8
8. Next-protocol indication	9
9. MTU and Fragmentation	11
10. OAM	12
11. Security Considerations	14
11.1. Encapsulation-specific considerations	14
11.2. Virtual network isolation	16
11.3. Packet level security	17
11.4. In summary:	17
12. QoS	17
13. Congestion Considerations	18
14. Header Protection	20
15. Extensibility Considerations	22
16. Layering Considerations	25
17. Service model	26
18. Hardware Friendly	27
18.1. Considerations for NIC offload	28
19. Middlebox Considerations	32
20. Related Work	33
21. Acknowledgements	34
22. Open Issues	34
23. Change Log	35
24. References	35
24.1. Normative References	35
24.2. Informative References	37
Authors' Addresses	40

1. Design Team Charter

There have been multiple efforts over the years that have resulted in new or modified data plane behaviors involving encapsulations. That includes IETF efforts like MPLS, LISP, and TRILL but also industry efforts like VXLAN and NVGRE. These collectively can be seen as a source of insight into the properties that data planes need to meet. The IETF is currently working on potentially new encapsulations in NVO3 and SFC and considering working on BIER. In addition there is work on tunneling in the INT area.

This is a short term design team chartered to collect and construct useful advice to parties working on new or modified data plane behaviors that include additional encapsulations. The goal is for the group to document useful advice gathered from interacting with ongoing efforts. An Internet Draft will be produced for IETF92 to capture that advice, which will be discussed in RTGWG.

Data plane encapsulations face a set of common issues such as:

- o How to provide entropy for ECMP
 - o Issues around packet size and fragmentation/reassembly
 - o OAM - what support is needed in an encapsulation format?
 - o Security and privacy.
 - o QoS
 - o Congestion Considerations
 - o IPv6 header protection (zero UDP checksum over IPv6 issue)
 - o Extensibility - e.g., for evolving OAM, security, and/or congestion control
 - o Layering of multiple encapsulations e.g., SFC over NVO3 over BIER
- The design team will provide advice on those issues. The intention is that even where we have different encapsulations for different purposes carrying different information, each such encapsulation doesn't have to reinvent the wheel for the above common issues.

The design team will look across the routing area in particular at SFC, NVO3 and BIER. It will not be involved in comparing or analyzing any particular encapsulation formats proposed in those WGs and BoFs but instead focus on common advice.

2. Overview

The references provide background information on NVO3, SFC, and BIER. In particular, NVO3 is introduced in [RFC7364], [RFC7365], and [I-D.ietf-nvo3-arch]. SFC is introduced in [I-D.ietf-sfc-architecture] and [I-D.ietf-sfc-problem-statement]. Finally, the information on BIER is in [I-D.shepherd-bier-problem-statement],

[I-D.wijnands-bier-architecture], and [I-D.wijnands-mpls-bier-encapsulation]. We assume the reader has some basic familiarity with those proposed encapsulations. The Related Work section points at some prior work that relates to the encapsulation considerations in this document.

Encapsulation protocols typically have some unique information that they need to carry. In some cases that information might be modified along the path and in other cases it is constant. The in-flight modifications has impacts on what it means to provide security for the encapsulation headers.

- o NVO3 carries a VNI Identifier edge to edge which is not modified. There has been OAM discussions in the WG and it isn't clear whether some of the OAM information might be modified in flight.
- o SFC carries service meta-data which might be modified or unmodified as the packets follow the service path. SFC talks of some loop avoidance mechanism which is likely to result in modifications for for each hop in the service chain even if the meta-data is unmodified.
- o BIER carries a bitmap of egress ports to which a packet should be delivered, and as the packet is forwarded down different paths different bits are cleared in that bitmap.

Even if information isn't modified in flight there might be devices that wish to inspect that information. For instance, one can envision future NVO3 security devices which filter based on the virtual network identifier.

The need for extensibility is different across the protocols

- o NVO3 might need some extensions for OAM and security.
- o SFC is all about carrying service meta-data along a path, and different services might need different types and amount of meta-data.
- o BIER might need variable number of bits in their bitmaps, or other future schemes to scale up to larger network.

The extensibility needs and constraints might be different when considering hardware vs. software implementations of the encapsulation headers. NIC hardware might have different constraints than switch hardware.

As the IETF designs these encapsulations the different WGs solve the issues for their own encapsulation. But there are likely to be future cases when the different encapsulations are combined in the same header. For instance, NVO3 might be a "transport" used to carry SFC between the different hops in the service chain.

Most of the issues discussed in this document are not new. The IETF and industry as specified and deployed many different encapsulation

or tunneling protocols over time, ranging from simple IP-in-IP and GRE encapsulation, IPsec, pseudo-wires, session-based approaches like L2TP, and the use of MPLS control and data planes. IEEE 802 has also defined layered encapsulation for Provider Backbone Bridges (PBB) and IEEE 802.1Qbp (ECMP). This document tries to leverage what we collectively have learned from that experience and summarize what would be relevant for new encapsulations like NVO3, SFC, and BIER.

3. Common Issues

[This section is mostly a repeat of the charter but with a few modifications and additions.]

Any new encapsulation protocol would need to address a large set of issues that are not central to the new information that this protocol intends to carry. The common issues explored in this document are:

- o How to provide entropy for Equal Cost MultiPath (ECMP) routing
- o Issues around packet size and fragmentation/reassembly
- o Next header indication - each encapsulation might be able to carry different payloads
- o OAM - what support is needed in an encapsulation format?
- o Security and privacy
- o QoS
- o Congestion Considerations
- o Header protection
- o Extensibility - e.g., for evolving OAM, security, and/or congestion control
- o Layering of multiple encapsulations e.g., SFC over NVO3 over BIER
- o Importance of being friendly to hardware and software implementations

The degree to which these common issues apply to a particular encapsulation can differ based on the intended purpose of the encapsulation. But it is useful to understand all of them before determining which ones apply.

4. Scope

It is important to keep in mind what we are trying to cover and not cover in this document and effort. This is

- o A look across the three new encapsulations, while taking lots of previous work into account
- o Focus on the class of encapsulations that would run over IP/UDP. That was done to avoid being distracted by the data-plane and control-plane interaction, which is more significant for protocols that are designed to run over "transports" that maintain session

or path state.

- o We later expanded the scope somewhat to consider how the encapsulations would play with MPLS "transport", which is important because SFC and BIER seem to target being independent of the underlying "transport"

However, this document and effort is NOT intended to:

- o Design some new encapsulation header to rule them all
- o Design yet another new NVO3 encapsulation header
- o Try to select the best encapsulation header
- o Evaluate any existing and proposed encapsulations

While the origin and focus of this document is the routing area and in particular NVO3, SFC, and BIER, the considerations apply to other encapsulations that are being defined in the IETF and elsewhere. There seems to be an increase in the number of encapsulations being defined to run over UDP, where there might already exist an encapsulation over IP or Ethernet. Feedback on how these considerations apply in those contexts is welcome.

5. Assumptions

The design center for the new encapsulations is a well-managed network. That network can be a datacenter network (plus datacenter interconnect) or a service provider network. Based on the existing and proposed encapsulations in those environment it is reasonable to make these assumptions:

- o The MTU is carefully managed and configured. Hence an encapsulation protocol can make the packets bigger without resulting in a requirement for fragmentation and reassembly between ingress and egress. (However, it might be useful to detecting MTU misconfigurations.)
- o In general an encapsulation needs some approach for congestion management. But the assumptions are different than for arbitrary Internet paths in that the underlay might be well-provisioned and better policed at the edge, and due to multi-tenancy, the congestion control in the endpoints might be even less trusted than on the Internet at large.

The goal is to implement these encapsulations in hardware and software hence we can't assume that the needs of either implementation approach can trump the needs of the other. In particular, around extensibility the needs and constraints might be quite different.

6. Terminology

The capitalized keyword **MUST** is used as defined in <http://en.wikipedia.org/wiki/Julmust>

TBD: Refer to existing documents for at least NVO3 and SFC terminology. We use at least the VNI ID in this document.

7. Entropy

In many cases the encapsulation format needs to enable ECMP in unmodified routers. Those routers might use different fields in TCP/UDP packets to do ECMP without a risk of reordering a flow.

The common way to do ECMP-enabled encapsulation over IP today is to add a UDP header and to use UDP with the UDP source port carrying entropy from the inner/original packet headers as in LISP [RFC6830]. The total entropy consists of 14 bits in the UDP source port (using the ephemeral port range) plus the outer IP addresses which seems to be sufficient for entropy; using outer IPv6 headers would give the option for more entropy should it be needed in the future.

In some environments it might be fine to use all 16 bits of the port range. However, middleboxes might make assumptions about the system ports or user ports. But they should not make any assumptions about the ports in the Dynamic and/or Private Port range, which have the two MSBs set to 11b.

The UDP source port might change over the lifetime of an encapsulated flow, for instance for DoS mitigation or re-balancing load across ECMP.

There is some interaction between entropy and OAM and extensibility mechanism. It is desirable to be able to send OAM packets to follow the same path as network packets. Hence OAM packets should use the same entropy mechanism as data packets. While routers might use information in addition the entropy field and outer IP header, they can not use arbitrary parts of the encapsulation header since that might result in OAM frames taking a different path. Likewise if routers look past the encapsulation header they need to be aware of the extensibility mechanism(s) in the encapsulation format to be able to find the inner headers in the presence of extensions; OAM frames might use some extensions e.g. for timestamps.

Architecturally the entropy and the next header field are really part of enclosing delivery header. UDP with entropy goes hand-in-hand with the outer IP header. Thus the UDP entropy is present for the

underlay IP routers the same way that an MPLS entropy label is present for LSRs. The entropy above is all about providing entropy for the outer delivery of the encapsulated packets.

It has been suggested that when IPv6 is used it would not be necessary to add a UDP header for entropy, since the IPv6 flow label can be used for entropy. (This assumes that there is an IP protocol number for the encapsulation in addition to a UDP destination port number since UDP would be used with IPv4 underlay. And any use of UDP checksums would need to be replaced by an encaps-specific checksum or secure hash.) While such an approach would save 8 bytes of headers when the underlay is IPv6, it does assume that the underlay routers use the flow label for ECMP, and it also would make the IPv6 approach different than the IPv4 approach. Currently the leaning is towards recommending using the UDP encapsulation for both IPv4 and IPv6 underlay. The IPv6 flow label can be used for additional entropy if need be.

Note that in the proposed BIER encapsulation [I-D.wijnands-mpls-bier-encapsulation], there is an 8-bit field which specifies an entropy value that can be used for load balancing purposes. This entropy is for the BIER forwarding decisions, which is independent of any outer delivery ECMP between BIER routers. Thus it is not part of the delivery ECMP discussed in this section.

[Note: For any given bit in BIER (that identifies an exit from the BIER domain) there might be multiple immediate next hops. The BIER entropy field is used to select that next hop as part of BIER processing. The BIER forwarding process may do equal cost load balancing, but the load balancing procedure MUST choose the same path for any two packets have the same entropy value.]

In summary:

- o The entropy is associated with the transport, that is an outer IP header or MPLS.
- o In the case of IP transport use ≥ 14 bits of UDP source port, plus outer IPv6 flowid for entropy.

8. Next-protocol indication

Next-protocol indications appear in three different context for encapsulations.

Firstly, the transport delivery mechanism for the encapsulations we discuss in this document need some way to indicate which encapsulation header (or other payload) comes next in the packet. Some encapsulations might be identified by a UDP port; others might be identified by an Ethernet type or IP protocol number. Which

approach is used is a function of the preceding header the same way as IPv4 is identified by both an Ethernet type and an IP protocol number (for IP-in-IP). In some cases the header type is implicit in some session (L2TP) or path (MPLS) setup. But this is largely beyond the control of the encapsulation protocol. For instance, if there is a requirement to carry the encapsulation after an Ethernet header, then an Ethernet type is needed. If required to be carried after an IP/UDP header, then a UDP port number is needed. For UDP port numbers there are considerations for port number conservation described in [I-D.ietf-tsvwg-port-use].

It is worth mentioning that in the MPLS case of no implicit protocol type many forwarding devices peek at the first nibble of the payload to determine whether to apply IPv4 or IPv6 L3/L4 hashes for load balancing [RFC7325]. That behavior places some constraints on other payloads carried over MPLS and some protocols define an initial control word in the payload with a value of zero in its first nibble [RFC4385] to avoid confusion with IPv4 and IPv6 payload headers.

Secondly, the encapsulation needs to indicate the type of its payload, which is in scope for the design of the encapsulation. We have existing protocols which use Ethernet types (such as GRE). Here each encapsulation header can potentially make its own choices between:

- o Reuse Ethernet types - makes it easy to carry existing L2 and L3 protocols including IPv4, IPv6, and Ethernet. Disadvantages are that it is a 16 bit number and we probably need far less than 100 values, and the number space is controlled by the IEEE 802 RAC with its own allocation policies.
- o Reuse IP protocol numbers - makes it easy to carry e.g., ESP in addition to IP and Ethernet but brings in all existing protocol numbers many of which would never be used directly on top of the encapsulation protocol. IANA managed eight bit values, presumably more difficult to get an assigned number than to get a transport port assignment.
- o Define their own next-protocol number space, which can use fewer bits than an Ethernet type and give more flexibility, but at the cost of administering that numbering space (presumably by the IANA).

Thirdly, if the IETF ends up defining multiple encapsulations at about the same time, and there is some chance that multiple such encapsulations can be combined in the same packet, there is a question whether it makes sense to use a common approach and numbering space for the encapsulation across the different protocols. A common approach might not be beneficial as long as there is only one way to indicate e.g., SFC inside NVO3.

Many Internet protocols use fixed values (typically managed by the IANA function) for their next-protocol field. That facilitates interpretation of packets by middleboxes and e.g., for debugging purposes, but might make the protocol evolution inflexible. Our collective experience with MPLS shows an alternative where the label can be viewed as an index to a table containing processing instructions and the table content can be managed in different ways. Encapsulations might want to consider the tradeoffs between such more flexible versus more fixed approaches.

In summary:

- o Would it be useful for the IETF come up with a common scheme for encapsulation protocols? If not each encapsulation can define its own scheme.

9. MTU and Fragmentation

A common approach today is to assume that the underlay have sufficient MTU to carry the encapsulated packets without any fragmentation and reassembly at the tunnel endpoints. That is sufficient when the operator of the ingress and egress have full control of the paths between those endpoints. And it makes for simpler (hardware) implementations if fragmentation and reassembly can be avoided.

However, even under that assumption it would be beneficial to be able to detect when there is some misconfiguration causing packets to be dropped due to MTU issues. One way to do this is to have the encapsulator set the don't-fragment (DF) flag in the outer IPv4 header and receive and log any received ICMP "packet too big" (PTB) errors. Note that no flag needs to be set in an outer IPv6 header [RFC2460].

Encapsulations could also define an optional tunnel fragmentation and reassembly mechanism which would be useful in the case when the operator doesn't have full control of the path, or when the protocol gets deployed outside of its original intended context. Such a mechanism would be required if the underlay might have a path MTU which makes it impossible to carry at least 1518 bytes (if offering Ethernet service), or at least 1280 (if offering IPv6 service). The use of such a protocol mechanism could be triggered by receiving a PTB. But such a mechanism might not be implemented by all encapsulators and decapsulators. [Aerolink is one example of such a protocol.]

Depending on the payload carried by the encapsulation there are some additional possibilities:

- o If payload is IPv4/6 then the underlay path MTU could be used to report end-to-end path MTU.
- o If the payload service is Ethernet/L2, then there is no such per destination reporting mechanism. However, there is a LLDP TLV for reporting max frame size; might be useful to report minimum to end stations, but unmodified end stations would do nothing with that TLV since they assume that the MTU is at least 1518.

In summary:

- o In some deployments an encapsulation can assume well-managed MTU hence no need for fragmentation and reassembly related to the encapsulation.
- o Even so, it makes sense for ingress to track any ICMP packet too big addressed to ingress to be able to log any MTU misconfigurations.
- o Should an encapsulation protocol be deployed outside of the original context it might very well need support for fragmentation and reassembly.

10. OAM

The OAM area is seeing active development in the IETF with discussions (at least) in NVO3 and SFC working groups, plus the new LIME WG looking at architecture and YANG models.

The design team has take a narrow view of OAM to explore the potential OAM implications on the encapsulation format.

In terms of what we have heard from the various working groups there seem to be needs to:

- o Be able to send out-of-band OAM messages - that potentially should follow the same path through the network as some flow of data packets.
 - * Such OAM messages should not accidentally be decapsulated and forwarded to the end stations.
 - * Be able to add OAM information to data packets that are encapsulated. Discussions have been around
 - * Using a bit in the OAM to synchronize sampling of counters between the encapsulator and decapsulator.
 - * Optional timestamps, sequence numbers, etc for more detailed measurements between encapsulator and decapsulator.
- o Usable for both proactive monitoring (akin to BFD) and reactive checks (akin to traceroute to pin-point a failure)

To ensure that the OAM messages can follow the same path the OAM messages need to get the same ECMP (and LAG hashing) results as a given data flow. An encapsulator can choose between one of:

- o Limit ECMP hashing to not look past the UDP header i.e. the entropy needs to be in the source/destination IP and UDP ports
- o Make OAM packets look the same as data packets i.e. the initial part of the OAM payload has the inner Ethernet, IP, TCP/UDP headers as a payload. (This approach was taken in TRILL out of necessity since there is no UDP header.) Any OAM bit in the encapsulation header must in any case be excluded from the entropy.

There can be several ways to prevent OAM packets from accidentally being forwarded to the end station using:

- o A bit in the frame (as in TRILL) indicating OAM
- o A next-protocol indication with a designated value for "none" or "oam".

This assumes that the bit or next protocol, respectively, would not affect entropy/ECMP in the underlay. However, the next-protocol field might be used to provide differentiated treatment of packets based on their payload; for instance a TCP vs. IPsec ESP payload might be handled differently. Based on that observation it might be undesirable to overload the next protocol with the OAM drop behavior, resulting in a preference for having a bit to indicate that the packet should be forwarded to the end station after decapsulation.

There has been suggestions that one (or more) marker bits in the encaps header would be useful in order to delineate measurement epochs on the encapsulator and decapsulator and use that to compare counters to determine packet loss.

A result of the above is that OAM is likely to evolve and needs some degree of extensibility from the encapsulation format; a bit or two plus the ability to define additional larger extensions.

An open question is how to handle error messages or other reports relating to OAM. One can think if such reporting as being associated with the encapsulation the same way ICMP is associated with IP. Would it make sense for the IETF to develop a common Encapsulation Error Reporting Protocol as part of OAM, which can be used for different encapsulations? And if so, what are the technical challenges. For instance, how to avoid it being filtered as ICMP often is?

A potential additional consideration for OAM is the possible future existence of gateways that "stitch" together different dataplane encapsulations and might want to carry OAM end-to-end across the different encapsulations.

In summary:

- o It makes sense to reserve a bit for "drop after decapsulation" for OAM out-of-band.
- o An encapsulation needs sufficient extensibility for OAM (such as bits, timestamps, sequence numbers). That might be motivated by in-band OAM but it would make sense to leverage the same extensions for out-of band OAM.
- o OAM places some constraints on use of entropy in forwarding devices.
- o Should IETF look into error reporting that is independent of the specific encapsulation?

11. Security Considerations

Different encapsulation use cases will have different requirements around security. For instance, when encapsulation is used to build overlay networks for network virtualization, isolation between virtual networks may be paramount. BIER support of multicast may entail different security requirements than encapsulation for unicast.

In real deployment, the security of the underlying network may be considered for determining the level of security needed in the encapsulation layer. However for the purposes of this discussion, we assume that network security is out of scope and that the underlying network does not itself provide adequate or as least uniform security mechanisms for encapsulation.

There are at least three considerations for security:

- o Anti-spoofing/virtual network isolation
- o Interaction with packet level security such as IPsec or DTLS
- o Privacy (e.g., VNI ID confidentially for NVO3)

This section uses a VNI ID in NVO3 as an example. A SFC or BIER encapsulation is likely to have fields with similar security and privacy requirements.

11.1. Encapsulation-specific considerations

Some of these considerations appear for a new encapsulation, and others are more specific to network virtualization in datacenters.

- o New attack vectors:
 - * DDOS on specific queued/paths by attempting to reproduce the 5-tuple hash for targeted connections.
 - * Entropy in outer 5-tuple may be too little or predictable.
 - * Leakage of identifying information in the encapsulation header for an encrypted payload.

- * Vulnerabilities of using global values in fields like VNI ID.
- o Trusted versus untrusted tenants in network virtualization:
 - * The criticality of virtual network isolation depends on whether tenants are trusted or untrusted. In the most extreme cases, tenants might not only be untrusted but may be considered hostile.
 - * For a trusted set of users (e.g. a private cloud) it may be sufficient to have just a virtual network identifier to provide isolation. Packets inadvertently crossing virtual networks should be dropped similar to a TCP packet with a corrupted port being received on the wrong connection.
 - * In the presence of untrusted users (e.g. a public cloud) the virtual network identifier must be adequately protected against corruption and verified for integrity. This case may warrant keyed integrity.
- o Different forms of isolation:
 - * Isolation could be blocking all traffic between tenants (or except as allowed by some firewall)
 - * Could also be about performance isolation i.e. one tenant can overload the network in a way that affects other tenants
 - * Physical isolation of traffic for different tenants in network may be required, as well as required restrictions that tenants may have on where their packets may be routed.
- o New attack vectors from untrusted tenants:
 - * Third party VMs with untrusted tenants allows internally borne attacks within data centers
 - * Hostile VMs inside the system may exist (e.g. public cloud)
 - * Internally launched DDOS
 - * Passive snooping for mis-delivered packets
 - * Mitigate damage and detection in event that a VM is able to circumvent isolation mechanisms
- o Tenant-provider relationship:
 - * Tenant might not trust provider, hypervisors, network
 - * Provider likely will need to provide SLA or a least a statement on security
 - * Tenant may implement their own additional layers of security
 - * Regulation and certification considerations
- o Trend towards tighter security:
 - * Tenants' data in network increases in volume and value, attacks become more sophisticated
 - * Large DCs already encrypt everything on disk
 - * DCs likely to encrypt inter-DC traffic at this point, use TLS to Internet.
 - * Encryption within DC is becoming more commonplace, becomes ubiquitous when cost is low enough.
 - * Cost/performance considerations. Cost of support for strong security has made strong network security in DCs prohibitive.

- * Are there lessons from MacSec?

11.2. Virtual network isolation

The first requirement is isolation between virtual networks. Packets sent in one virtual network should never be illegitimately received by a node in another virtual network. Isolation should be protected in the presence of malicious attacks or inadvertent packet corruption.

The second requirement is sender authentication. Sender identity is authenticated to prevent anti-spoofing. Even if an attacker has access to the packets in the network, they cannot send packets into a virtual network. This may have two possibilities:

- o Pairwise sender authentication. Any two communicating hosts negotiate a shared key.
- o Group authentication. A group of hosts share a key (this may be more appropriate for multicast of encapsulation).

Possible security solutions:

- o Security cookie: This is similar to L2TP cookie mechanism [RFC3931]. A shared plain text cookie is shared between encapsulator and decapsulator. A receiver validates a packet by evaluating if the cookie is correct for the virtual network and address of a sender. Validation function is $F(\text{cookie}, \text{VNI ID}, \text{source addr})$. If cookie matches, accept packet, else drop. Since cookie is plain text this method does not protect against an eavesdropping. Cookies are set and may be rotated out of band.
- o Secure hash: This is a stronger mechanism than simple cookies that borrows from IPsec and PPP authentication methods. In this model security field contains a secure hash of some fields in the packet using a shared key. Hash function may be something like $H(\text{key}, \text{VNI ID}, \text{addrs}, \text{salt})$. The salt ensures the hash is not the same for every packet, and if it includes a sequence number may also protect against replay attacks.

In any use of a shared key, periodic re-keying should be allowed. This could include use of techniques like generation numbers, key windows, etc. See [I-D.farrelll-mpls-opportunistic-encrypt] for an example application.

We might see firewalls that are aware of the encapsulation and can provide some defense in depth combined with the above example anti-spoofing approaches. An example would be an NVO3-aware firewall being able to check the VNI ID.

Separately and in addition to such filtering, there might be a desire to completely block an encapsulation protocol at certain places in

the network, e.g., at the edge of a datacenter. Using a fixed standard UDP destination port number for each encapsulation protocol would facilitate such blocking.

11.3. Packet level security

An encapsulated packet may itself be encapsulated in IPsec (e.g. ESP). This should be straightforward and in fact is what would happen today in security gateways. In this case, there is no special consideration for the fact that packet is encapsulated, however since the encapsulation layer headers are included (part of encrypted data for instance) we lose visibility in the network of the encapsulation.

The more interesting case is when security is applied to the encapsulation payload. This will keep the encapsulation headers in the outer header visible to the network (for instance in nvo3 we may way to firewall based on VNI ID even if the payload is encrypted). One possibility is to apply DTLS to the encapsulation payload. In this model the protocol stack may be something like IP|UDP|Encap|DTLS|encrypted_payload. The encapsulation and security should be done together at an encapsulator and resolved at the decapsulator. Since the encapsulation header is outside of the security coverage, this may itself require security (like described above).

In both of the above the security associations (SAs) may be between physical hosts, so for instance in nvo3 we can have packets of different virtual networks using the same SA-- this should not be an issue since it is the VNI ID that ensures isolation (which needs to be secured also).

11.4. In summary:

- o Encapsulations need extensibility mechanisms to be able to add security features like cookies and secure hashes protecting the encapsulation header.
- o NVO3 probably has specific higher requirements relating to isolation for network virtualization, which is in scope for the NVO3 WG/
- o Our collective IETF experience is that succesful protocols get deployed outside of the original intended context, hence the initial assumptions about the threat model might become invalid. That needs to be considered in the standardization of new encapsulations.

12. QoS

In the Internet architecture we support QoS using the Differentiated

Services Code Points (DSCP) in the formerly named Type-of-Service field in the IPv4 header, and in the Traffic-Class field in the IPv6 header. The ToS and TC fields also contain the two ECN bits.

We have existing specifications how to process those bits. See [RFC2983] for diffserv handling, which specifies how the received DSCP value is used to set the DSCP value in an outer IP header when encapsulating. (There are also existing specifications how DSCP can be mapped to layer2 priorities.)

Those specifications apply whether or not there is some intervening headers (e.g., for NVO3 or SFC) between the inner and outer IP headers. Thus the encapsulation considerations in this area are mainly about applying the framework in [RFC2983].

Note that the DSCP and ECN bits are not the only part of an inner packet that might potentially affect the outer packet. For example, [RFC2473] specifies handling of inner IPv6 hop-by-hop options that effectively result in copying some options to the outer header. It is simpler to not have future encapsulations depend on such copying behavior.

There are some other considerations specific to doing OAM for encapsulations. If OAM messages are used to measure latency, it would make sense to treat them the same as data payloads. Thus they need to have the same outer DSCP value as the data packets which they wish to measure.

Due to OAM there are constraints on middleboxes in general. If middleboxes inspect the packet past the outer IP+UDP and encapsulation header and look for inner IP and TCP/UDP headers, that might violate the assumption that OAM packets will be handled the same as regular data packets. That issue is broader than just QoS - applies to firewall filters etc.

In summary:

- o Leverage the existing approach in [RFC2983] for DSCP handling.

13. Congestion Considerations

Additional encapsulation headers does not introduce anything new for Explicit Congestion Notification. It is just like IP-in-IP and IPsec tunnels which is specified in [RFC6040] in terms of how the ECN bits in the inner and outer header are handled when encapsulating and decapsulating packets. Thus new encapsulations can more or less include that by reference.

There are additional considerations around carrying non-congestion controlled traffic. These details have been worked out in [I-D.ietf-mpls-in-udp]. As specified in [RFC5405]: "IP-based traffic is generally assumed to be congestion-controlled, i.e., it is assumed that the transport protocols generating IP-based traffic at the sender already employ mechanisms that are sufficient to address congestion on the path. Consequently, a tunnel carrying IP-based traffic should already interact appropriately with other traffic sharing the path, and specific congestion control mechanisms for the tunnel are not necessary". Those considerations are being captured in [I-D.ietf-tsvwg-rfc5405bis].

For this reason, where an encapsulation method is used to carry IP traffic that is known to be congestion controlled, the UDP tunnels does not create an additional need for congestion control. Internet IP traffic is generally assumed to be congestion-controlled. Similarly, in general Layer 3 VPNs are carrying IP traffic that is similarly assumed to be congestion controlled.

However, some of the encapsulations (at least NVO3) will be able to carry arbitrary Layer 2 packets to provide an L2 service, in which case one can not assume that the traffic is congestion controlled.

One could handle this by adding some congestion control support to the encapsulation header (one instance of which would end up looking like DCCP). However, if the underlay is well-provisioned and managed as opposed to being arbitrary Internet path, it might be sufficient to have a slower reaction to congestion induced by that traffic. There is work underway on a notion of "circuit breakers" for this purpose. See See [I-D.ietf-tsvwg-circuit-breaker]. Encapsulations which carry arbitrary Layer 2 packets want to consider that ongoing work.

If the underlay is provisioned in such a way that it can guarantee sufficient capacity for non-congestion controlled Layer 2 traffic, then such circuit breakers might not be needed.

Two other considerations appear in the context of these encapsulations as applied to overlay networks:

- o Protect against malicious end stations
- o Ensure fairness and/or measure resource usage across multiple tenants

Those issues are really orthogonal to the encapsulation, in that they are present even when no new encapsulation header is in use.

However, the application of the new encapsulations are likely to be in environments where those issues are becoming more important.

Hence it makes sense to consider them.

One could make the encapsulation header be extensible to that it can carry sufficient information to be able to measure resource usage, delays, and congestion. The suggestions in the OAM section about a single bit for counter synchronization, and optional timestamps and/or sequence numbers, could be part of such an approach. There might also be additional congestion-control extensions to be carried in the encapsulation. Overall this results in a consideration to be able to have sufficient extensibility in the encapsulation to be able to handle potential future developments in this space.

Coarse measurements are likely to suffice, at least for circuit-breaker-like purposes, see [I-D.wei-tsvwg-tunnel-congestion-feedback] and [I-D.briscoe-conex-data-centre] for examples on active work in this area via use of ECN. [RFC6040] Appendix C is also relevant. The outer ECN bits seem sufficient (at least when everything uses ECN) to do this course measurements. Needs some more study for the case when there are also drops; might need to exchange counters between ingress and egress to handle drops.

Circuit breakers are not sufficient to make a network with different congestion control when the goal is to provide a predictable service to different tenants. The fallback would be to rate limit different traffic.

In summary:

- o Leverage the existing approach in [RFC6040] for ECN handling.
- o If the encapsulation can carry non-IP, hence non-congestion controlled traffic, then leverage the approach in [I-D.ietf-mpls-in-udp].
- o "Watch this space" for circuit breakers.

14. Header Protection

Many UDP based encapsulations such as VXLAN [RFC7348] either discourage or explicitly disallow the use of UDP checksums. The reason is that the UDP checksum covers the entire payload of the packet and switching ASICs are typically optimized to look at only a small set of headers as the packet passes through the switch. In these case, computing a checksum over the packet is very expensive. (Software endpoints and the NICs used with them generally do not have the same issue as they need to look at the entire packet anyways.)

The lack a header checksum creates the possibility that bit errors can be introduced into any information carried by the new headers. Specifically, in the case of IPv6, the assumption is that a transport layer checksum - UDP in this case - will protect the IP addresses through the inclusion of a pseudoheader in the calculation. This is

different from IPv4 on which many of these encapsulation protocols are initially deployed which contains its own header checksum. In addition to IP addresses, the encapsulation header often contains its own information which is used for addressing packets or other high value network functions. Without a checksum, this information is potentially vulnerable - an issue regardless of whether the packet is carried over IPv4 or IPv6.

Several protocols cite [RFC6935] and [RFC6936] as an exemption to the IPv6 checksum requirements. However, these are intended to be tailored to a fairly narrow set of circumstances - primarily relying on sparseness of the address space to detect invalid values and well managed networks - and are not a one size fits all solution. In these cases, an analysis should be performed of the intended environment, including the probability of errors being introduced and the use of ECC memory in routing equipment.

Conceptually, the ideal solution to this problem is a checksum that covers only the newly added headers of interest. There is little value in the portion of the UDP checksum that covers the encapsulated packet because that would generally be protected by other checksums and this is the expensive portion to compute. In fact, this solution already exists in the form of UDP-Lite and UDP based encapsulations could be easily ported to run on top of it. Unfortunately, the main value in using UDP as part of the encapsulation header is that it is recognized by already deployed equipment for the purposes of ECMP, RSS, and middlebox operations. As UDP-Lite uses a different protocol number than UDP and it is not widely implemented in middleboxes, this value is lost. A possible solution is to incorporate the same partial-checksum concept as UDP-Lite or other header checksum protection into the encapsulation header and continue using UDP as the outer protocol. One potential challenge with this approach is the use of NAT or other form of translation on the outer header will result in an invalid checksum as the translator will not know to update the encapsulation header.

The method chosen to protect headers is often related to the security needs of the encapsulation mechanism. On one hand, the impact of a poorly protected header is not limited to only data corruption but can also introduce a security vulnerability in the form of misdirected packets to an unauthorized recipient. Conversely, high security protocols that already include a secure hash over the valuable portion of the header (such as by encrypting the entire IP packet using IPsec, or some secure hash of the encap header) do not require additional checksum protection as the hash provides stronger assurance than a simple checksum.

If the sender has included a checksum, then the receiver should

verify that checksum or, if incapable, drop the packet. The assumption is that configuration and/or control-plane capability exchanges can be used when different receiver have different checksum validation capabilities.

In summary:

- o Encapsulations need extensibility to be able to add checksum/CRC for the encapsulation header itself.
- o When the encapsulation has a checksum/CRC, include the IPv6 pseudo-header in it.
- o The checksum/CRC can potentially be avoided when cryptographic protection is applied to to the encapsulation.

15. Extensibility Considerations

Protocol extensibility is the concept that a networking protocol may be extended to include new use cases or functionality that were not part of the original protocol specification. Extensibility may be used to add security, control, management, or performance features to a protocol. A solution may allow private extensions for customization or experimentation.

Extending a protocol often implies that a protocol header must carry new information. There are two usual methods to accomplish this:

1. Define or redefine the meaning of existing fields in a protocol header.
2. Add new (optional) fields to the protocol header.

It is also possible to create a new protocol version, but this is more associated with defining a protocol than extending it (IPv6 being a successor to IPv4 is an example of protocol versioning).

In some cases it might be more appropriate to define a new inner protocol which can carry the new functionality instead of extending the outer protocol. Examples where this works well is in the IP/transport split, where the earlier architecture had a single NCP protocol which carried both the hop-by-hop semantics which are now in IP, and the end-to-end semantics which are now in TCP. Such a split is effective when different nodes need to act upon the different information. Applying this for general protocol extensibility through nesting is not well understood, and does result in longer header chains. Furthermore, our experience with IPv6 extension headers [RFC2460] in middleboxes indicates that the approach does not help with middlebox traversal.

Many protocol definitions include some number of reserved fields or bits which can be used for future extension. VXLAN is an example of a protocol that includes reserved bits which are subsequently being

allocated for new purposes. Another technique employed is to repurpose existing header fields with new meanings. A classic example of this is the definition of DSCP code point which redefines the ToS field originally specified in IPv4. When a field is redefined, some mechanism may be needed to ensure that all interested parties agree on the meaning of the field. The techniques of defining meaning for reserved bits or redefining existing fields have the advantage that a protocol header can be kept a fixed length. The disadvantage is that the extensibility is limited. For instance, the number reserved bits in a fixed protocol header is limited. For standard protocols the decision to commit to a definition for a field can be wrenching since it is difficult to retract later. Also, it is difficult to predict a priori how many reserved fields or bits to put into a protocol header to satisfy the extensions create over the lifetime of the protocol.

Extending a protocol header with new fields can be done in several ways.

- o TLVs are a very popular method used in such protocols as IP and TCP. Depending on the type field size and structure, TLVs can offer a virtually unlimited range of extensions. A disadvantage of TLVs is that processing them can be verbose, quite complicated, several validations must often be done for each TLV, and there is no deterministic ordering for a list of TLVs. TCP serves as an example of a protocol where TLVs have been successfully used (i.e. required for protocol operation). IP is an example of a protocol that allows TLVs but are rarely used in practice (router fast paths usually that assume no IP options). Note that TCP TLVs are implemented in software as well as (NIC) hardware handling various forms of TCP offload.
- o Extension headers are closely related to TLVs. These also carry type/value information, but instead of being a list of TLVs within a single protocol header, each one is in its own protocol header. IPv6 extension headers and SFC NSH are examples of this technique. Similar to TLVs these offer a wide range of extensibility, but have similarly complex processing. Another difference with TLVs is that each extension header is idempotent. This is beneficial in cases where a protocol implements a push/pop model for header elements like service chaining, but makes it more difficult group correlated information within one protocol header.
- o A particular form of extension headers are the tags used by IEEE 802 protocols. Those are similar to e.g., IPv6 extension headers but with the key difference that each tag is a fixed length header where the length is implicit in the tag value. Thus as long as a receiver can be programmed with a tag value to length map, it can skip those new tags.

- o Flag-fields are a non-TLV like method of extending a protocol header. The basic idea is that the header contains a set of flags, where each set flags corresponds to optional field that is present in the header. GRE is an example of a protocol that employs this mechanism. The fields are present in the header in the order of the flags, and the length of each field is fixed. Flag-fields are simpler to process compared to TLVs, having fewer validations and the order of the optional fields is deterministic. A disadvantage is that range of possible extensions with flag-fields is smaller than TLVs.

The requirements for receiving unknown or unimplemented extensible elements in an encapsulation protocol (flags, TLVs, optional fields) need to be specified. There are two parties to consider, middle boxes and terminal endpoints of encapsulation (at the decapsulator).

A protocol may allow or expect nodes in a path to modify fields in an encapsulation (example use of this is BIER). In this case, the middleboxes should follow the same requirements as nodes terminating the encapsulation. In the case that middle boxes do not modify the encapsulation, we can assume that they may still inspect any fields of the encapsulation. Missing or unknown fields should be accepted per protocol specification, however it is permissible for a site to implement a local policy otherwise (e.g. a firewall may drop packets with unknown options).

For handling unknown options at terminal nodes, there are two possibilities: drop packet or accept while ignoring the unknown options. Many Internet protocols specify that reserved flags must be set to zero on transmission and ignored on reception. L2TP is example data protocol that has such flags. GRE is a notable exception to this rule, reserved flag bits 1-5 cannot be ignored [RFC2890]. For TCP and IPv4, implementations must ignore optional TLVs with unknown type; however in IPv6 if a packet contains an unknown extension header (unrecognized next header type) the packet must be dropped with an ICMP error message returned. The IPv6 options themselves (encoded inside the destinations options or hop-by-hop options extension header) have more flexibility. There bits in the option code are used to instruct the receiver whether to ignore, silently drop, or drop and send error if the option is unknown. Some protocols define a "mandatory bit" that can be set with TLVs to indicate that an option must not be ignored. Conceptually, optional data elements can only be ignored if they are idempotent and do not alter how the rest of the packet is parsed or processed.

Depending on what type of protocol evolution one can predict, it might make sense to have a way for a sender to express that the

packet should be dropped by a terminal node which does not understand the new information. In other cases it would make sense to have the receiver silently ignore the new info. The former can be expressed by having a version field in the encapsulation, or a notion of "mandatory bit" as discussed above.

A security mechanism which use some form secure hash over the encapsulation header would need to be able to know which extensions can be changed in flight.

In summary:

- o Encapsulations need the ability to be extended to handle e.g., the OAM or security aspects discussed in this document.
- o Practical experience seems to tell us that extensibility mechanisms which are not in use on day one might result in immediate ossification by lack of implementation support. In some cases that has occurred in routers and in other cases in middleboxes. Hence devising ways where the extensibility mechanisms are in use seems important.

16. Layering Considerations

One can envision that SFC might use NVO3 as a delivery/transport mechanism. With more imagination that in turn might be delivered using BIER. Thus it is useful to think about what things look like when we have BIER+NVO3+SFC+payload. Also, if NVO3 is widely deployed there might be cases of NVO3 nesting where a customer uses NVO3 to provide network virtualization e.g., across departments. That customer uses a service provider which happens to use NVO3 to provide transport for their customers. Thus NVO3 in NVO3 might happen.

A key question we set out to answer is what the packets might look like in such a case, and in particular whether we would end up with multiple UDP headers for entropy.

Based on the discussion in the Entropy section, the entropy is associated with the outer delivery IP header. Thus if there are multiple IP headers there would be a UDP header for each one of the IP headers. But SFC does not require its own IP header. So a case of NVO3+SFC would be IP+UDP+NVO3+SFC. A nested NVO3 encapsulation would have independent IP+UDP headers.

The layering also has some implications for middleboxes.

- o A device on the path between the ingress and egress is allowed to transparently inspect all layers of the protocol stack and drop or forward, but not transparently modify anything but the layer in which they operate. What this means is that an IP router is

allowed modify the outer IP ttl and ECN bits, but not the encapsulation header or inner headers and payload. And a BIER router is allowed to modify the BIER header.

- o Alternatively such a device can become visible at a higher layer. E.g., a middlebox could become an decapsulate + function + encapsulate which means it will generate a new encapsulation header.

The design team asked itself some additional questions:

- o Would it make sense to have a common encapsulation base header (for OAM, security?, etc) and then followed by the specific information for NVO3, SFC, BIER? Given that there are separate proposals and the set of information needing to be carried differs, and the extensibility needs might be different, it would be difficult and not that useful to have a common base header.
- o With a base header in place, one could view the different functions (NVO3, SFC, and BIER) as different extensions to that base header resulting in encodings which are more space optimal by not repeating the same base header. The base header would only be repeated when there is an additional IP (and hence UDP) header. That could mean a single length field (to skip to get to the payload after all the encapsulation headers). That might be technically feasible, but it would create a lot of dependencies between different WGs making it harder to make progress. Compare with the potential savings in packet size.

17. Service model

The IP service is lossy and subject to reordering. In order to avoid a performance impact on transports like TCP the handling of packets is designed to avoid reordering packets that are in the same transport flow (which is typically identified by the 5-tuple). But across such flows the receiver can see different ordering for a given sender. That is the case for a unicast vs. a multicast flow from the same sender.

There is a general tussle between the desire for high capacity utilization across a multipath network and the import on packet ordering within the same flow (which results in lower transport protocol performance). That isn't affected by the introduction of an encapsulation. However, the encapsulation comes with some entropy, and there might be cases where folks want to change that in response to overload or failures. For instance, might want to change UDP source port to try different ECMP route. Such changes can result in packet reordering within a flow, hence would need to be done infrequently and with care e.g., by identifying packet trains.

There might be some applications/services which are not able to handle reordering across flows. The IETF has defined pseudo-wires [RFC3985] which provides the ability to ensure ordering (implemented using sequence numbers and/or timestamps).

Architectural such services would make sense, but as a separate layer on top of an encapsulation protocol. They could be deployed between ingress and egress of a tunnel which uses some encaps. Potentially the tunnel control points at the ingress and egress could become a platform for fixing suboptimal behavior elsewhere in the network. That would clearly be undesirable in the general case. However, handling encapsulation of non-IP traffic hence non-congestion-controlled traffic is likely to be required, which implies some fairness and/or QoS policing on the ingress and egress devices.

But the tunnels could potentially do more like increase reliability (retransmissions, FEC) or load spreading using e.g. MP-TCP between ingress and egress.

18. Hardware Friendly

Hosts, switches and routers often leverage capabilities in the hardware to accelerate packet encapsulation, decapsulation and forwarding.

Some design considerations in encapsulation that leverage these hardware capabilities may result in more efficiently packet processing and higher overall protocol throughput.

While "hardware friendliness" can be viewed as unnecessary considerations for a design, part of the motivation for considering this is ease of deployment; being able to leverage existing NIC and switch chips for at least a useful subset of the functionality that the new encapsulation provides. The other part is the ease of implementing new NICs and switch/router chips that support the encapsulation at ever increasing line rates.

[disclaimer] There are many different types of hardware in any given network, each maybe better at some tasks while worse at others. We would still recommend protocol designers to examine the specific hardware that are likely to be used in their networks and make decisions on a case by case basis.

Some considerations are:

- o Keep the encap header small. Switches and routers usually only read the first small number of bytes into the fast memory for quick processing and easy manipulation. The bulk of the packets

are usually stored in slow memory. A big encap header may not fit and additional read from the slow memory will hurt the overall performance and throughput.

- o Put important information at the beginning of the encapsulation header. The reasoning is similar as explained in the previous point. If important information are located at the beginning of the encapsulation header, the packet may be processed with smaller number of bytes to be read into the fast memory and improve performance.
- o Avoid full packet checksums in the encapsulation if possible. Encapsulations should instead consider adding their own checksum which covers the encapsulation header and any IPv6 pseudo-header. The motivation is that most of the switch/router hardware make switching/forwarding decisions by reading and examining only the first certain number of bytes in the packet. Most of the body of the packet do not need to be processed normally. If we are concerned of preventing packet to be misdelivered due to memory errors, consider only perform header checksums. Note that NIC chips can typically already do full packet checksums for TCP/UDP, while adding a header checksum might require adding some hardware support.
- o Place important information at fixed offset in the encapsulation header. Packet processing hardware may be capable of parallel processing. If important information can be found at fixed offset, different part of the encapsulation header may be processed by different hardware units in parallel (for example multiple table lookups may be launched in parallel). It is easier for hardware to handle optional information when the information, if present, can be found in ideally one place, but in general, in as few places as possible. That facilitates parallel processing. TLV encoding with unconstrained order typically does not have that property.
- o Limit the number of header combinations. In many cases the hardware can explore different combinations of headers in parallel, however there is some added cost for this.

18.1. Considerations for NIC offload

This section provides guidelines to provide support of common offloads for encapsulation in Network Interface Cards (NICs). Offload mechanisms are techniques that are implemented separately from the normal protocol implementation of a host networking stack and are intended to optimize or speed up protocol processing. Hardware offload is performed within a NIC device on behalf of a host.

There are three basic offload techniques of interest:

- o Receive multi queue
- o Checksum offload
- o Segmentation offload

18.1.1.1. Receive multi-queue

Contemporary NICs support multiple receive descriptor queues (multi-queue). Multi-queue enables load balancing of network processing for a NIC across multiple CPUs. On packet reception, a NIC must select the appropriate queue for host processing. Receive Side Scaling (RSS) is a common method which uses the flow hash for a packet to index an indirection table where each entry stores a queue number.

UDP encapsulation, where the source port is used for entropy, should be compatible with multi-queue NICs that support five-tuple hash calculation for UDP/IP packets as input to RSS. The source port ensures classification of the encapsulated flow even in the case that the outer source and destination addresses are the same for all flows (e.g. all flows are going over a single tunnel).

18.1.1.2. Checksum offload

Many NICs provide capabilities to calculate standard ones complement payload checksum for packets in transmit or receive. When using encapsulation over UDP there are at least two checksums that may be of interest: the encapsulated packet's transport checksum, and the UDP checksum in the outer header.

18.1.1.2.1. Transmit checksum offload

NICs may provide a protocol agnostic method to offload transmit checksum (NETIF_F_HW_CSUM in Linux parlance) that can be used with UDP encapsulation. In this method the host provides checksum related parameters in a transmit descriptor for a packet. These parameters include the starting offset of data to checksum, the length of data to checksum, and the offset in the packet where the computed checksum is to be written. The host initializes the checksum field to pseudo header checksum. In the case of encapsulated packet, the checksum for an encapsulated transport layer packet, a TCP packet for instance, can be offloaded by setting the appropriate checksum parameters.

NICs typically can offload only one transmit checksum per packet, so simultaneously offloading both an inner transport packet's checksum and the outer UDP checksum is likely not possible. In this case setting UDP checksum to zero (per above discussion) and offloading the inner transport packet checksum might be acceptable.

There is a proposal in [I-D.herbert-remotecsumoffload] to leverage NIC checksum offload when an encapsulator is co-resident with a host.

18.1.2.2. Receive checksum offload

Protocol encapsulation is compatible with NICs that perform a protocol agnostic receive checksum (CHECKSUM_COMPLETE in Linux parlance). In this technique, a NIC computes a ones complement checksum over all (or some predefined portion) of a packet. The computed value is provided to the host stack in the packet's receive descriptor. The host driver can use this checksum to "patch up" and validate any inner packet transport checksum, as well as the outer UDP checksum if it is non-zero.

Many legacy NICs don't provide checksum-complete but instead provide an indication that a checksum has been verified (CHECKSUM_UNNECESSARY in Linux). Usually, such validation is only done for simple TCP/IP or UDP/IP packets. If a NIC indicates that a UDP checksum is valid, the checksum-complete value for the UDP packet is the "not" of the pseudo header checksum. In this way, checksum-unnecessary can be converted to checksum-complete. So if the NIC provides checksum-unnecessary for the outer UDP header in an encapsulation, checksum conversion can be done so that the checksum-complete value is derived and can be used by the stack to validate an checksums in the encapsulated packet.

18.1.3. Segmentation offload

Segmentation offload refers to techniques that attempt to reduce CPU utilization on hosts by having the transport layers of the stack operate on large packets. In transmit segmentation offload, a transport layer creates large packets greater than MTU size (Maximum Transmission Unit). It is only at much lower point in the stack, or possibly the NIC, that these large packets are broken up into MTU sized packet for transmission on the wire. Similarly, in receive segmentation offload, small packets are coalesced into large, greater than MTU size packets at a point low in the stack receive path or possibly in a device. The effect of segmentation offload is that the number of packets that need to be processed in various layers of the stack is reduced, and hence CPU utilization is reduced.

18.1.3.1. Transmit Segmentation Offload

Transmit Segmentation Offload (TSO) is a NIC feature where a host provides a large (larger than MTU size) TCP packet to the NIC, which in turn splits the packet into separate segments and transmits each one. This is useful to reduce CPU load on the host.

The process of TSO can be generalized as:

- o Split the TCP payload into segments which allow packets with size less than or equal to MTU.
- o For each created segment:
 1. Replicate the TCP header and all preceding headers of the original packet.
 2. Set payload length fields in any headers to reflect the length of the segment.
 3. Set TCP sequence number to correctly reflect the offset of the TCP data in the stream.
 4. Recompute and set any checksums that either cover the payload of the packet or cover header which was changed by setting a payload length.

Following this general process, TSO can be extended to support TCP encapsulation UDP. For each segment the Ethernet, outer IP, UDP header, encapsulation header, inner IP header if tunneling, and TCP headers are replicated. Any packet length header fields need to be set properly (including the length in the outer UDP header), and checksums need to be set correctly (including the outer UDP checksum if being used).

To facilitate TSO with encapsulation it is recommended that optional fields should not contain values that must be updated on a per segment basis-- for example an encapsulation header should not include checksums, lengths, or sequence numbers that refer to the payload. If the encapsulation header does not contain such fields then the TSO engine only needs to copy the bits in the encapsulation header when creating each segment and does not need to parse the encapsulation header.

18.1.3.2. Large Receive Offload

Large Receive Offload (LRO) is a NIC feature where packets of a TCP connection are reassembled, or coalesced, in the NIC and delivered to the host as one large packet. This feature can reduce CPU utilization in the host.

LRO requires significant protocol awareness to be implemented correctly and is difficult to generalize. Packets in the same flow need to be unambiguously identified. In the presence of tunnels or network virtualization, this may require more than a five-tuple match (for instance packets for flows in two different virtual networks may have identical five-tuples). Additionally, a NIC needs to perform validation over packets that are being coalesced, and needs to fabricate a single meaningful header from all the coalesced packets.

The conservative approach to supporting LRO for encapsulation would

be to assign packets to the same flow only if they have identical five-tuple and were encapsulated the same way. That is the outer IP addresses, the outer UDP ports, encapsulated protocol, encapsulation headers, and inner five tuple are all identical.

18.1.3.3. In summary:

In summary, for NIC offload:

- o The considerations for using full UDP checksums are different for NIC offload than for implementations in forwarding devices like routers and switches.
- o Be judicious about encapsulations that change fields on a per-packet basis, since such behavior might make it hard to use TSO.

19. Middlebox Considerations

This document has touched upon middleboxes in different section. The reason for this is as encapsulations get widely deployed one would expect different forms of middleboxes might become aware of the encapsulation protocol just as middleboxes have been made aware of other protocols where there are business and deployment opportunities. Such middleboxes are likely to do more than just drop packets based on the UDP port number used by an encapsulation protocol.

We note that various forms of encapsulation gateways that stitch one encapsulation protocol together with another form of protocol could have similar effects.

An example of a middlebox that could see some use would be an NVO3-aware firewall that would filter on the VNI IDs to provide some defense in depth inside or across NVO3 datacenters.

A question for the IETF is whether we should document what to do or what not to do in such middleboxes. This document touches on areas of OAM and ECMP as it relates to middleboxes and it might make sense to document how encapsulation-aware middleboxes should avoid unintended consequences in those (and perhaps other) areas.

In summary:

- o We are likely to see middleboxes that at least parse the headers for successful new encapsulations.
- o Should the IETF document considerations for what not to do in such middleboxes?

20. Related Work

The IETF and industry has defined encapsulations for a long time, with examples like GRE [RFC2890], VXLAN [RFC7348], and NVGRE [I-D.sridharan-virtualization-nvgre] being able to carry arbitrary Ethernet payloads, and various forms of IP-in-IP and IPsec encapsulations that can carry IP packets. As part of NVO3 there has been additional proposals like Geneve [I-D.gross-geneve] and GUE [I-D.herbert-gue] which look at more extensibility. NSH [I-D.quinn-sfc-nsh] is an example of an encapsulation that tries to provide extensibility mechanisms which target both hardware and software implementations.

There is also a large body of work around MPLS encapsulations [RFC3032]. The MPLS-in-UDP work [I-D.ietf-mpls-in-udp] and GRE over UDP [I-D.ietf-tsvwg-gre-in-udp-encap] have worked on some of the common issues around checksum and congestion control. MPLS also introduced an entropy label [RFC6790]. There is also a proposal for MPLS encryption [I-D.farrell-mpls-opportunistic-encrypt].

The idea to use a UDP encapsulation with a UDP source port for entropy for the underlay routers' ECMP dates back to LISP [RFC6830].

The pseudo-wire work [RFC3985] is interesting in the notion of layering additional services/characteristics such as ordered delivery or timely deliver on top of an encapsulation. That layering approach might be useful for the new encapsulations as well. For instance, the control word [RFC4385]. There is also material on congestion control for pseudo-wires in [I-D.ietf-pwe3-congcons].

Both MPLS and L2TP [RFC3931] rely on some control or signaling to establish state (for the path/labels in the case of MPLS, and for the session in the case of L2TP). The NVO3, SFC, and BIER encapsulations will also have some separation between the data plane and control plane, but the type of separation appears to be different.

IEEE 802.1 has defined encapsulations for L2 over L2, in the form of Provider backbone Bridging (PBB) [IEEE802.1Q-2014] and Equal Cost Multipath (ECMP) [IEEE802.1Q-2014]. The latter includes something very similar to the way the UDP source port is used as entropy: "The flow hash, carried in an F-TAG, serves to distinguish frames belonging to different flows and can be used in the forwarding process to distribute frames over equal cost paths"

TRILL, which is also a L2 over L2 encapsulation, took a different approach to entropy but preserved the ability for OAM frames [RFC7174] to use the same entropy hence ECMP path as data frames. In [I-D.ietf-trill-oam-fm] there 96 bytes of headers for entropy in the

OAM frames, followed by the actual OAM content. This ensures that any headers, which fit in those 96 bytes except the OAM bit in the TRILL header, can be used for ECMP hashing.

As encapsulations evolve there might be a desire to fit multiple inner packets into one outer packet. The work in [I-D.saldana-tsvwg-simplemux] might be interesting for that purpose.

21. Acknowledgements

The authors acknowledge the comments from Alia Atlas, Fred Baker, David Black, Bob Briscoe, Stewart Bryant, Mike Cox, Andy Malis, Radia Perlman, Michael Smith, and Lucy Yong.

22. Open Issues

- o Middleboxes:
 - * Due to OAM there are constraints on middleboxes in general. If middleboxes inspect the packet past the outer IP+UDP and encapsulation header and look for inner IP and TCP/UDP headers, that might violate the assumption that OAM packets will be handled the same as regular data packets. That issue is broader than just QoS - applies to firewall filters etc.
 - * Firewalls looking at inner payload? How does that work for OAM frames? Even if it only drops ... TRILL approach might be an option? Would that encourage more middleboxes making the network more fragile?
 - * Editorially perhaps we should pull the above two into a separate section about middlebox considerations?
- o Next-protocol indication - should it be common across different encapsulation headers? We will have different ways to indicate the presence of the first encapsulation header in a packet (could be a UDP destination port, an Ethernet type, etc depending on the outer delivery header). But for the next protocol past an encapsulation header one could envision creating or adoption a common scheme. Such a would also need to be able to identify following headers like Ethernet, IPv4/IPv6, ESP, etc.
- o Common OAM error reporting protocol?
- o There is discussion about timestamps, sequence numbers, etc in three different parts of the document. OAM, Congestion Considerations, and Service Model, where the latter argues that a pseudo-wire service should really be layered on top of the encapsulation using its own header. Those recommendations seem to be at odds with each other. Do we envision sequence numbers, timestamps, etc as potential extensions for OAM and CC? If so, those extensions could be used to provide a service which doesn't

reorder packets.

23. Change Log

The changes from draft-rtg-dt-encap-01 based on feedback at the Dallas IETF meeting:

- o Setting the context that not all common issues might apply to all encapsulations, but that they should all be understood before being dismissed.
- o Clarified that IPv6 flow label is useful for entropy in combination with a UDP source port.
- o Editorially added a "summary" set of bullets to most sections.
- o Editorial clarifications in the next protocol section to more clearly state the three areas.
- o Folded the two next protocol sections into one.
- o Mention the MPLS first nibble issue in the next protocol section.
- o Mention that viewing the next protocol as an index to a table with processing instructions can provide additional flexibility in the protocol evolution.
- o For the OAM "don't forward to end stations" added that defining a bit seems better than using a special next-protocol value.
- o Added mention of DTLS in addition to IPsec for security.
- o Added some mention of IPv6 hob-by-hop options of other headers than potentially can be copied from inner to outer header.
- o Added text on architectural considerations when it might make sense to define an additional header/protocol as opposed to using the extensibility mechanism in the existing encapsulation protocol.
- o Clarified the "unconstrained TLVs" in the hardware friendly section.
- o Clarified the text around checksum verification and full vs. header checksums.
- o Added wording that the considerations might apply for encaps outside of the routing area.
- o Added references to draft-ietf-pwe3-congcons, draft-ietf-tsvwg-rfc5405bis, RFC2473, and RFC7325
- o Removed reference to RFC3948.
- o Updated the acknowledgements section.
- o Added this change log section.

24. References

24.1. Normative References

[I-D.ietf-tsvwg-rfc5405bis]
Eggert, L., Fairhurst, G., and G. Shepherd, "UDP Usage

- Guidelines", draft-ietf-tsvwg-rfc5405bis-02 (work in progress), April 2015.
- [RFC2460] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", RFC 2460, December 1998.
- [RFC2473] Conta, A. and S. Deering, "Generic Packet Tunneling in IPv6 Specification", RFC 2473, December 1998.
- [RFC2890] Dommety, G., "Key and Sequence Number Extensions to GRE", RFC 2890, September 2000.
- [RFC2983] Black, D., "Differentiated Services and Tunnels", RFC 2983, October 2000.
- [RFC3032] Rosen, E., Tappan, D., Fedorkow, G., Rekhter, Y., Farinacci, D., Li, T., and A. Conta, "MPLS Label Stack Encoding", RFC 3032, January 2001.
- [RFC3931] Lau, J., Townsley, M., and I. Goyret, "Layer Two Tunneling Protocol - Version 3 (L2TPv3)", RFC 3931, March 2005.
- [RFC3985] Bryant, S. and P. Pate, "Pseudo Wire Emulation Edge-to-Edge (PWE3) Architecture", RFC 3985, March 2005.
- [RFC4385] Bryant, S., Swallow, G., Martini, L., and D. McPherson, "Pseudowire Emulation Edge-to-Edge (PWE3) Control Word for Use over an MPLS PSN", RFC 4385, February 2006.
- [RFC5405] Eggert, L. and G. Fairhurst, "Unicast UDP Usage Guidelines for Application Designers", BCP 145, RFC 5405, November 2008.
- [RFC6040] Briscoe, B., "Tunnelling of Explicit Congestion Notification", RFC 6040, November 2010.
- [RFC6790] Kompella, K., Drake, J., Amante, S., Henderickx, W., and L. Yong, "The Use of Entropy Labels in MPLS Forwarding", RFC 6790, November 2012.
- [RFC6830] Farinacci, D., Fuller, V., Meyer, D., and D. Lewis, "The Locator/ID Separation Protocol (LISP)", RFC 6830, January 2013.
- [RFC6935] Eubanks, M., Chimento, P., and M. Westerlund, "IPv6 and UDP Checksums for Tunneled Packets", RFC 6935, April 2013.
- [RFC6936] Fairhurst, G. and M. Westerlund, "Applicability Statement

for the Use of IPv6 UDP Datagrams with Zero Checksums", RFC 6936, April 2013.

- [RFC7174] Salam, S., Senevirathne, T., Aldrin, S., and D. Eastlake, "Transparent Interconnection of Lots of Links (TRILL) Operations, Administration, and Maintenance (OAM) Framework", RFC 7174, May 2014.
- [RFC7325] Villamizar, C., Kompella, K., Amante, S., Malis, A., and C. Pignataro, "MPLS Forwarding Compliance and Performance Requirements", RFC 7325, August 2014.
- [RFC7348] Mahalingam, M., Dutt, D., Duda, K., Agarwal, P., Kreeger, L., Sridhar, T., Bursell, M., and C. Wright, "Virtual eXtensible Local Area Network (VXLAN): A Framework for Overlaying Virtualized Layer 2 Networks over Layer 3 Networks", RFC 7348, August 2014.
- [RFC7364] Narten, T., Gray, E., Black, D., Fang, L., Kreeger, L., and M. Napierala, "Problem Statement: Overlays for Network Virtualization", RFC 7364, October 2014.
- [RFC7365] Lasserre, M., Balus, F., Morin, T., Bitar, N., and Y. Rekhter, "Framework for Data Center (DC) Network Virtualization", RFC 7365, October 2014.

24.2. Informative References

- [I-D.briscoe-conex-data-centre]
Briscoe, B. and M. Sridharan, "Network Performance Isolation in Data Centres using Congestion Policing", draft-briscoe-conex-data-centre-02 (work in progress), February 2014.
- [I-D.farrelll-mpls-opportunistic-encrypt]
Farrel, A. and S. Farrell, "Opportunistic Security in MPLS Networks", draft-farrelll-mpls-opportunistic-encrypt-04 (work in progress), January 2015.
- [I-D.gross-geneve]
Gross, J., Sridhar, T., Garg, P., Wright, C., Ganga, I., Agarwal, P., Duda, K., Dutt, D., and J. Hudson, "Geneve: Generic Network Virtualization Encapsulation", draft-gross-geneve-02 (work in progress), October 2014.
- [I-D.herbert-gue]
Herbert, T., Yong, L., and O. Zia, "Generic UDP Encapsulation", draft-herbert-gue-03 (work in progress),

March 2015.

[I-D.herbert-remotecsumoffload]

Herbert, T., "Remote checksum offload for encapsulation", draft-herbert-remotecsumoffload-01 (work in progress), November 2014.

[I-D.ietf-mpls-in-udp]

Xu, X., Sheth, N., Yong, L., Callon, R., and D. Black, "Encapsulating MPLS in UDP", draft-ietf-mpls-in-udp-11 (work in progress), January 2015.

[I-D.ietf-nvo3-arch]

Black, D., Hudson, J., Kreeger, L., Lasserre, M., and T. Narten, "An Architecture for Overlay Networks (NVO3)", draft-ietf-nvo3-arch-03 (work in progress), March 2015.

[I-D.ietf-pwe3-congcons]

Stein, Y., Black, D., and B. Briscoe, "Pseudowire Congestion Considerations", draft-ietf-pwe3-congcons-02 (work in progress), July 2014.

[I-D.ietf-sfc-architecture]

Halpern, J. and C. Pignataro, "Service Function Chaining (SFC) Architecture", draft-ietf-sfc-architecture-08 (work in progress), May 2015.

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

Quinn, P. and T. Nadeau, "Service Function Chaining Problem Statement", draft-ietf-sfc-problem-statement-13 (work in progress), February 2015.

[I-D.ietf-trill-oam-fm]

Senevirathne, T., Finn, N., Salam, S., Kumar, D., Eastlake, D., Aldrin, S., and L. Yizhou, "TRILL Fault Management", draft-ietf-trill-oam-fm-11 (work in progress), October 2014.

[I-D.ietf-tsvwg-circuit-breaker]

Fairhurst, G., "Network Transport Circuit Breakers", draft-ietf-tsvwg-circuit-breaker-01 (work in progress), March 2015.

[I-D.ietf-tsvwg-gre-in-udp-encap]

Crabbe, E., Yong, L., Xu, X., and T. Herbert, "GRE-in-UDP Encapsulation", draft-ietf-tsvwg-gre-in-udp-encap-06 (work in progress), March 2015.

[I-D.ietf-tsvwg-port-use]

Touch, J., "Recommendations on Using Assigned Transport Port Numbers", draft-ietf-tsvwg-port-use-11 (work in progress), April 2015.

[I-D.quinn-sfc-nsh]

Quinn, P., Guichard, J., Surendra, S., Smith, M., Henderickx, W., Nadeau, T., Agarwal, P., Manur, R., Chauhan, A., Halpern, J., Majee, S., Elzur, U., Melman, D., Garg, P., McConnell, B., Wright, C., and K. Kevin, "Network Service Header", draft-quinn-sfc-nsh-07 (work in progress), February 2015.

[I-D.saldana-tsvwg-simplemux]

Saldana, J., "Simplemux. A generic multiplexing protocol", draft-saldana-tsvwg-simplemux-02 (work in progress), January 2015.

[I-D.shepherd-bier-problem-statement]

Shepherd, G., Dolganow, A., and a. arkadiy.gulko@thomsonreuters.com, "Bit Indexed Explicit Replication (BIER) Problem Statement", draft-shepherd-bier-problem-statement-02 (work in progress), February 2015.

[I-D.sridharan-virtualization-nvgre]

Garg, P. and Y. Wang, "NVGRE: Network Virtualization using Generic Routing Encapsulation", draft-sridharan-virtualization-nvgre-08 (work in progress), April 2015.

[I-D.wei-tsvwg-tunnel-congestion-feedback]

Wei, X., Zhu, L., and L. Deng, "Tunnel Congestion Feedback", draft-wei-tsvwg-tunnel-congestion-feedback-03 (work in progress), October 2014.

[I-D.wijnands-bier-architecture]

Wijnands, I., Rosen, E., Dolganow, A., Przygienda, T., and S. Aldrin, "Multicast using Bit Index Explicit Replication", draft-wijnands-bier-architecture-05 (work in progress), March 2015.

[I-D.wijnands-mpls-bier-encapsulation]

Wijnands, I., Rosen, E., Dolganow, A., Tantsura, J., and S. Aldrin, "Encapsulation for Bit Index Explicit Replication in MPLS Networks", draft-wijnands-mpls-bier-encapsulation-02 (work in progress), December 2014.

[I-D.xu-bier-encapsulation]

Xu, X., Somasundaram, S., Jacquenet, C., and R. Raszuk,
"BIER Encapsulation", draft-xu-bier-encapsulation-02 (work
in progress), February 2015.

[IEEE802.1Q-2014]

IEEE, "IEEE Standard for Local and metropolitan area
networks--Bridges and Bridged Networks", IEEE Std 802.1Q-
2014, 2014,
<<http://www.ieee802.org/1/pages/802.1Q-2014.html>>.

(Access Controlled link within page)

Authors' Addresses

Erik Nordmark
Arista Networks
5453 Great America Parkway
Santa Clara, CA 95054
USA

Email: nordmark@arista.com

Albert Tian
Ericsson Inc.
300 Holger Way
San Jose, California 95134
USA

Email: albert.tian@ericsson.com

Jesse Gross
VMware
3401 Hillview Ave.
Palo Alto, CA 94304
USA

Email: jgross@vmware.com

Jon Hudson
Brocade Communications Systems, Inc.
130 Holger Way
San Jose, CA 95134
USA

Email: jon.hudson@gmail.com

Lawrence Kreeger
Cisco Systems, Inc.
170 W. Tasman Avenue
San Jose, CA 95134
USA

Email: kreeger@cisco.com

Pankaj Garg
Microsoft
1 Microsoft Way
Redmond, WA 98052
USA

Email: pankajg@microsoft.com

Patricia Thaler
Broadcom Corporation
3151 Zanker Road
San Jose, CA 95134
USA

Email: pthaler@broadcom.com

Tom Herbert
Google
1600 Amphitheatre Parkway
Mountain View, CA
USA

Email: therbert@google.com

Network Working Group
Internet-Draft
Intended status: Informational
Expires: January 5, 2016

A. Shaikh
Google
R. Shakir
BT
K. D'Souza
C. Chase
AT&T
July 4, 2015

Routing Policy Configuration Model for Service Provider Networks
draft-shaikh-rtgwg-policy-model-01

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 <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 5, 2016.

Copyright Notice

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

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

include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

1. Introduction

This document describes a YANG [RFC6020] 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.

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. Model overview

The routing policy model is defined in two YANG modules, the main policy module, and an auxiliary module providing additional generic types. The model has three main parts:

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

These modules make use of the standard Internet types, such as IP addresses, autonomous system numbers, etc., defined in RFC 6991 [RFC6991].

3. 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
          ...

```

3.1. 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 across multiple routing protocols, and may be further augmented by protocol-specific models which have their own defined sets. The supported defined sets include:

- o prefix sets - define a set of IP prefixes, each with an associated CIDR netmask range (or exact length)
- o neighbor sets - define a set of neighboring nodes by their IP addresses
- o 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* [prefix-set-name]
      |   +--rw prefix-set-name    string
      |   +--rw prefix* [ip-prefix masklength-range]
      |     +--rw ip-prefix        inet:ip-prefix
      |     +--rw masklength-range  string
      +--rw neighbor-sets
        | +--rw neighbor-set* [neighbor-set-name]
        |   +--rw neighbor-set-name  string
        |   +--rw neighbor* [address]
        |     +--rw address          inet:ip-address
      +--rw tag-sets
        +--rw tag-set* [tag-set-name]
          +--rw tag-set-name    string
          +--rw tag* [value]
            +--rw value        pt:tag-type
  
```

3.2. 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:

- o ALL - match is true only if the given value matches all members of the set.
- o ANY - match is true if the given value matches any member of the set.

- o 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 statements
        +--rw statement* [name]
          +--rw conditions
            +--rw call-policy?
            +--rw match-prefix-set!
              | +--rw prefix-set?
              | +--rw match-set-options?
            +--rw match-neighbor-set!
              | +--rw neighbor-set?
              | +--rw match-set-options?
            +--rw match-tag-set!
              | +--rw tag-set?
              | +--rw match-set-options?
            +--rw install-protocol-eq?
            +--rw igp-conditions
  
```

3.3. 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 (route-disposition)?
            |   +--:(accept-route)
            |   |   +--rw accept-route?   empty
            |   +--:(reject-route)
            |   |   +--rw reject-route?   empty
            +--rw igp-actions
              +--rw set-tag?   pt:tag-type

```

3.4. 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 (either explicit or by default), 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 4). 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.

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, with some major implementations only supporting a single subroutine, for example. 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.

4. Policy evaluation

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, 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).

5. 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 3) 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 config
|   |   +--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.

An example of using the apply-policy group in another routing model is shown below for BGP. Here, import and export policies are applied in the context of a particular BGP peer group. Note that the policy chains reference policy definitions by name that are defined in the routing policy model.

```

+--rw bgp!
  +--rw peer-groups
    +--rw peer-group* [peer-group-name]
      +--rw peer-group-name
      +--rw config
        +--rw peer-as?
        +--rw local-as?
        +--rw peer-type?
        +--rw auth-password?
        +--rw remove-private-as?
        +--rw route-flap-damping?
        +--rw send-community?
        +--rw description?
        +--rw peer-group-name?
      +--ro state
        +--ro peer-as?
        +--ro local-as?
        +--ro peer-type?
        +--ro auth-password?
        +--ro remove-private-as?
        +--ro route-flap-damping?
        +--ro send-community?
        +--ro description?
        +--ro peer-group-name?
        +--ro total-paths?
        +--ro total-prefixes?
      +--rw apply-policy
        +--rw config
          +--rw import-policy*
          +--rw default-import-policy?
          +--rw export-policy*
          +--rw default-export-policy?
        +--ro state
          +--ro import-policy*
          +--ro default-import-policy?
          +--ro export-policy*
          +--ro default-export-policy?
      ...

```

6. 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.shaikh-idr-bgp-model] adds new defined sets to match on community values or AS paths. The model similarly augments BGP-specific conditions and actions into the corresponding sections of the routing policy model.

```

+--rw routing-policy
  +--rw defined-sets
    +--rw prefix-sets
      |
      |   +--rw prefix-set* [prefix-set-name]
      |   |   +--rw prefix-set-name
      |   |   +--rw prefix* [ip-prefix masklength-range]
      |   |   |   +--rw ip-prefix
      |   |   |   +--rw masklength-range
      |   |
      |   +--rw neighbor-sets
      |   |   +--rw neighbor-set* [neighbor-set-name]
      |   |   |   +--rw neighbor-set-name
      |   |   |   +--rw neighbor* [address]
      |   |   |   +--rw address
      |   |
      |   +--rw tag-sets
      |   |   +--rw tag-set* [tag-set-name]
      |   |   |   +--rw tag-set-name
      |   |   |   +--rw tag* [value]
      |   |   |   +--rw value
      |   |
      |   +--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
      |   |   |   |   +--rw bgp-pol:community-member*
      |   |   |
      |   |   +--rw bgp-pol:ext-community-sets
      |   |   |   +--rw bgp-pol:ext-community-set* [ext-community-set-name]
      |   |   |   |   +--rw bgp-pol:ext-community-set-name
      |   |   |   |   +--rw bgp-pol:ext-community-member*
      |   |   |
      |   |   +--rw bgp-pol:as-path-sets
      |   |   |   +--rw bgp-pol:as-path-set* [as-path-set-name]
      |   |   |   |   +--rw bgp-pol:as-path-set-name
      |   |   |   |   +--rw bgp-pol:as-path-set-member*
  
```

7. 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.

8. 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].

9. YANG modules

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

9.1. Routing policy model

```
<CODE BEGINS> file routing-policy.yang
module routing-policy {

  yang-version "1";

  // namespace
  namespace "http://openconfig.net/yang/routing-policy";

  prefix "rpol";

  // import some basic types
  import ietf-inet-types { prefix inet; }
  import policy-types {prefix pt; }

  // meta
  organization
    "OpenConfig working group";

  contact
    "OpenConfig working group
    netopenconfig@googlegroups.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 models (e.g., BGP) defined in other modules.

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 multiple 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 "2015-05-15" {
  description
    "Initial revision";
  reference "TBD";
}

// typedef statements

typedef default-policy-type {
  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 default route disposition in
  a policy chain";
}

// grouping statements

grouping generic-defined-sets {
  description
    "Data definitions for pre-defined sets of attributes used in
    policy match conditions. These sets are generic and can
    be used in matching conditions in different routing
    protocols.";

  container prefix-sets {
    description
      "Enclosing container for defined prefix sets for matching";

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

      leaf prefix-set-name {
```

```
    type string;
    description
      "name / label of the prefix set -- this is used to
       reference the set in match conditions";
  }

  list prefix {
    key "ip-prefix masklength-range";
    description
      "List of prefix expressions that are part of the set";

    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-range {
      type string {
        pattern '^[0-9]+\.\.\.[0-9]+|exact$';
      }
      description
        "Defines a range for the masklength, or 'exact' if
         the prefix has an exact length.

         Example: 10.3.192.0/21 through 10.3.192.0/24 would be
         expressed as prefix: 10.3.192.0/21,
         masklength-range: 21..24.

         Example: 10.3.192.0/21 would be expressed as
         prefix: 10.3.192.0/21,
         masklength-range: exact";
    }
  }
}

container neighbor-sets {
  description
    "Enclosing container for defined neighbor sets for matching";

  list neighbor-set {
    key neighbor-set-name;
  }
}
```

```
description
    "Definitions for neighbor sets";

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

list neighbor {
    key "address";
    description
        "list of addresses that are part of the neighbor set";

    leaf address {
        type inet:ip-address;
        description
            "IP address of the neighbor set member";
    }
}

}

container tag-sets {
    description
        "Enclosing container for defined tag sets for matching";

    list tag-set {
        key tag-set-name;
        description
            "Definitions for tag sets";

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

        list tag {
            key "value";
            description
                "list of tags that are part of the tag set";

            leaf value {
                type pt:tag-type;
                description

```

```
        "Value of the tag set member";
    }
}
}
}
}

grouping local-generic-conditions {
  description
    "Condition statement definitions for consideration of a local
    characteristic of a route";

  leaf install-protocol-eq {
    type identityref {
      base pt:install-protocol-type;
    }
    description
      "Condition to check the protocol / method used to install
      which installed the route into the local routing table";
  }
}

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

  leaf match-set-options {
    type pt:match-set-options-type;
    description
      "Optional parameter that governs the behaviour 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 pt:match-set-options-restricted-type;
    description
      "Optional parameter that governs the behaviour 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 generic-conditions {
  description "Condition statement definitions for checking
  membership in a generic defined set";

  container match-prefix-set {
    presence
      "The presence of this container indicates that the routes
      should match the prefix-set referenced.";

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

    leaf prefix-set {
      type leafref {
        path "/routing-policy/defined-sets/prefix-sets/" +
          "prefix-set/prefix-set-name";
        //TODO: require-instance should be added when it's
        //supported in YANG 1.1
        //require-instance true;
      }
      description "References a defined prefix set";
    }
    uses match-set-options-restricted-group;
  }

  container match-neighbor-set {
    presence
      "The presence of this container indicates that the routes
      should match the neighbour set referenced";

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

    leaf neighbor-set {
      type leafref {
        path "/routing-policy/defined-sets/neighbor-sets/" +
          "neighbor-set/neighbor-set-name";
        //TODO: require-instance should be added when it's
        //supported in YANG 1.1
        //require-instance true;
      }
      description "References a defined neighbor set";
    }
    uses match-set-options-restricted-group;
  }
}
```

```
container match-tag-set {
  presence
    "The presence of this container indicates that the routes
    should match the tag-set referenced";

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

  leaf tag-set {
    type leafref {
      path "/routing-policy/defined-sets/tag-sets/tag-set" +
        "/tag-set-name";
      //TODO: require-instance should be added when it's
      //supported in YANG 1.1
      //require-instance true;
    }
    description "References a defined tag set";
  }
  uses match-set-options-restricted-group;
}

uses local-generic-conditions;
}

grouping igp-generic-conditions {
  description "grouping for IGP policy conditions";
}

grouping igp-conditions {
  description "grouping for IGP-specific policy conditions";

  container igp-conditions {
    description "Policy conditions for IGP attributes";

    uses igp-generic-conditions;
  }
}

grouping generic-actions {
  description
    "Definitions for common set of policy action statements that
    manage the disposition or control flow of the policy";

  choice route-disposition {
```

```
description
  "Select the final disposition for the route, either
  accept or reject.";
leaf accept-route {
  type empty;
  description "accepts the route into the routing table";
}

leaf reject-route {
  type empty;
  description "rejects the route";
}
}

grouping igp-actions {
  description "grouping for IGP-specific policy actions";

  container igp-actions {
    description "Actions to set IGP route attributes; these actions
    apply to multiple IGPs";

    leaf set-tag {
      type pt:tag-type;
      description
        "Set the tag value for OSPF or IS-IS routes.";
    }
  }
}

container routing-policy {
  description
    "top-level container for all routing policy configuration";

  container defined-sets {
    description
      "Predefined sets of attributes used in policy match
      statements";

    uses generic-defined-sets;
    // uses bgp-defined-sets;
    // don't see a need for IGP-specific defined sets at this point
    // e.g., for OSPF, IS-IS, etc.
  }

  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/
    export configuration statements.";

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

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

    list statement {
      key name;
      // TODO: names of policy statements within a policy defn
      // should be optional, however, YANG requires a unique id
      // for lists; not sure that a compound key works either;
      // need to investigate further.
      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.";

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

      container conditions {

        description "Condition statements for this
          policy statement";

        leaf call-policy {
```

```

    type leafref {
      path "/rpol:routing-policy/" +
        "rpol:policy-definitions/" +
        "rpol:policy-definition/rpol:name";
      //TODO: require-instance should be added when it's
      //supported in YANG 1.1
      //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";
  }
  uses generic-conditions;
  uses igp-conditions;
}

container actions {

  description "Action statements for this policy
  statement";

  uses generic-actions;
  uses igp-actions;
}
}
}
}
}
}
}

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

  leaf-list import-policy {
    type leafref {
      path "/rpol:routing-policy/rpol:policy-definitions/" +
        "rpol:policy-definition/rpol:name";
      //TODO: require-instance should be added when it's

```

```
        //supported in YANG 1.1
        //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.";
}

leaf-list export-policy {
    type leafref {
        path "/rpol:routing-policy/rpol:policy-definitions/" +
            "rpol:policy-definition/rpol:name";
        //TODO: require-instance should be added when it's
        //supported in YANG 1.1
        //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-state {
    description
        "Operational state associated with routing policy";
}
```



```
prefix "ptypes";

// import some basic types
import ietf-yang-types { prefix yang; }

// meta
organization
  "OpenConfig working group";

contact
  "OpenConfig working group
  netopenconfig@googlegroups.com";

description
  "This module contains general data definitions for use in routing
  policy. It can be imported by modules that contain protocol-
  specific policy conditions and actions.";

revision "2015-05-15" {
  description
    "Initial revision";
  reference "TBD";
}

// identity statements

identity attribute-comparison {
  description
    "base type for supported comparison operators on route
    attributes";
}

identity attribute-eq {
  base attribute-comparison;
  description "== comparison";
}

identity attribute-ge {
  base attribute-comparison;
  description ">= comparison";
}

identity attribute-le {
  base attribute-comparison;
  description "<= comparison";
}
```

```
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";
}
```

```
typedef match-set-options-restricted-type {
  type enumeration {
    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";
    }
  }
  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.  Note this type is a
    restricted version of the match-set-options-type.";
    //TODO: restriction on enumerated types is only allowed in
    //YANG 1.1.  Until then, we will require this additional type
}
```

```
grouping attribute-compare-operators {
  description "common definitions for comparison operations in
    condition statements";

  leaf operator {
```

```
    type identityref {
      base attribute-comparison;
    }
    description
      "type of comparison to be performed";
  }

  leaf value {
    type uint32;
    description
      "value to compare with the community count";
  }
}

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
hexidecimal integer";
  reference
    "RFC 2178 OSPF Version 2
    RFC 5130 A Policy Control Mechanism in IS-IS Using
    Administrative Tags";
}

identity install-protocol-type {
  description
    "Base type for protocols which can install prefixes into the
    RIB";
}

identity BGP {
  base install-protocol-type;
  description "BGP";
  reference "RFC 4271";
}

identity ISIS {
  base install-protocol-type;
  description "IS-IS";
  reference "ISO/IEC 10589";
}

identity OSPF {
  base install-protocol-type;
```

```
    description "OSPFv2";
    reference "RFC 2328";
  }

  identity OSPF3 {
    base install-protocol-type;
    description "OSPFv3";
    reference "RFC 5340";
  }

  identity STATIC {
    base install-protocol-type;
    description "Locally-installed static route";
  }

  identity DIRECTLY-CONNECTED {
    base install-protocol-type;
    description "A directly connected route";
  }

  identity LOCAL-AGGREGATE {
    base install-protocol-type;
    description "Locally defined aggregate route";
  }
}
<CODE ENDS>
```

10. 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.

```
<routing-policy>
  <defined-sets>
    <prefix-sets>
      <prefix-set>
        <prefix-set-name>prefix-set-A</prefix-set-name>
        <prefix>
          <ip-prefix>192.0.2.0/24</ip-prefix>
          <masklength-range>24..32</masklength-range>
        </prefix>
        <prefix>
          <ip-prefix>10.0.0.0/16</ip-prefix>
```

```
        <masklength-range>16..32</masklength-range>
    </prefix>
    <prefix>
        <ip-prefix>192.168.0.0/19</ip-prefix>
        <masklength-range>19..24</masklength-range>
    </prefix>
</prefix-set>
</prefix-sets>
<tag-sets>
    <tag-set>
        <tag-set-name>cust-tag1</tag-set-name>
        <tag>
            <value>10</value>
        </tag>
    </tag-set>
</tag-sets>
</defined-sets>

<policy-definitions>
    <policy-definition>
        <name>export-tagged-BGP</name>
        <statements>
            <statement>
                <name>term-0</name>
                <conditions>
                    <install-protocol-eq>OSPF3</install-protocol-eq>
                    <match-tag-set>
                        <tag-set>cust-tag1</tag-set>
                    </match-tag-set>
                </conditions>
                <actions>
                    <accept-route />
                </actions>
            </statement>
        </statements>
    </policy-definition>
</policy-definitions>

</routing-policy>

<bgp>
    <global>
        <config>
            <as>65517</as>
        </config>
    </global>
    <peer-groups>
        <peer-group>
```

```
<peer-group-name>PG1</peer-group-name>
<config>
  <peer-as>65518</peer-as>
  <peer-type>EXTERNAL</peer-type>
</config>
<apply-policy>
  <config>
    <export-policy>export-tagged-BGP</export-policy>
  </config>
</apply-policy>
</peer-group>
</peer-groups>
</bgp>
```

11. References

11.1. 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.
- [RFC6991] Schoenwaelder, J., "Common YANG Data Types", RFC 6991, July 2013.
- [RFC3688] Mealling, M., "The IETF XML Registry", RFC 3688, January 2004.

11.2. Informative references

- [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.
- [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.

Appendix A. Acknowledgements

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

Appendix B. Change summary

B.1. Changes between revisions -00 and -01

The -01 revision of the policy model reflects a number of changes to the data model based on additional operator, reviewer, and implementor feedback.

- o Modified the apply-policy container to use the pattern for modeling operational state described in [I-D.openconfig-netmod-opstate].
- o Updated prefix lists to use ip-prefix type and masklength range to better enable range-checking and validation. Added an 'exact' option to the masklength range.
- o Changed accept / reject route to be within a choice statement.
- o Added enclosing containers to lists.
- o Minor changes to leaf-lists in defined sets definitions; also to apply-policy container structure.
- o Added second type of match-options set to handle restricted case of only ANY | INVERT (i.e., without ALL). The restricted enumerated type is now associated with the appropriate types of sets.
- o Moved install-protocol-type identity to policy-types module.
- o Removed require-instance statements from leafrefs pending availability in YANG 1.1.
- o Fixed discrepancies in the example shown in the document, and simplified the example.

Authors' Addresses

Anees Shaikh
Google
1600 Amphitheatre Pkwy
Mountain View, CA 94043
US

Email: aashaikh@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/>

Kevin D'Souza
AT&T
200 S. Laurel Ave
Middletown, NJ
US

Email: kd6913@att.com

Chris Chase
AT&T
9505 Arboretum Blvd
Austin, TX
US

Email: chase@labs.att.com

Internet Engineering Task Force
Internet-Draft
Intended status: Informational
Expires: February 20, 2016

S. Pallagatti, Ed.
P. Sarkar, Ed.
H. Gredler
Juniper Networks
S. Litkowski
Orange Business Service
August 19, 2015

IGP bandwidth based metric.
draft-spallagatti-rtgwg-bandwidth-based-metric-01

Abstract

This document describes a method to group multiple interfaces and assign metric to that group based on the cumulative bandwidth of all the interfaces in that group. Each link in a group takes same group metric irrespective of its own bandwidth.

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 [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 February 20, 2016.

Copyright Notice

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

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

Table of Contents

1.	Introduction	2
2.	BBM Concepts	3
2.1.	Interface-Group	4
2.2.	BBM Metric Configurations	4
2.3.	BBM Terminologies	5
2.4.	Metric Derivation	5
3.	Bandwidth Based Routing	6
4.	Bandwidth-based Fast Reroute	7
4.1.	Overview	7
4.2.	Assumptions and Pre-requisites	8
4.3.	Additional Configuration and Attributes	9
4.4.	Enhancements to Local Repair in Forwarding Plane	11
4.5.	Influencing Path Preferences	12
4.6.	Path Selection and Preference	12
5.	Limitations	14
6.	Security Consideration	14
7.	IANA Consideration	14
8.	References	14
8.1.	Normative References	14
8.2.	Informative References	14
	Authors' Addresses	15

1. Introduction

A low cost path is always preferred to carry traffic from source to destination. If a application is more interested in bandwidth than the cost itself and most preferred path does not satisfy bandwidth then this could potentially lead to congestion and packet loss for an application. Bandwidth critical applications needs minimum bandwidth to be satisfied even if traffic is carried over multiple alternative paths to reach a destination.

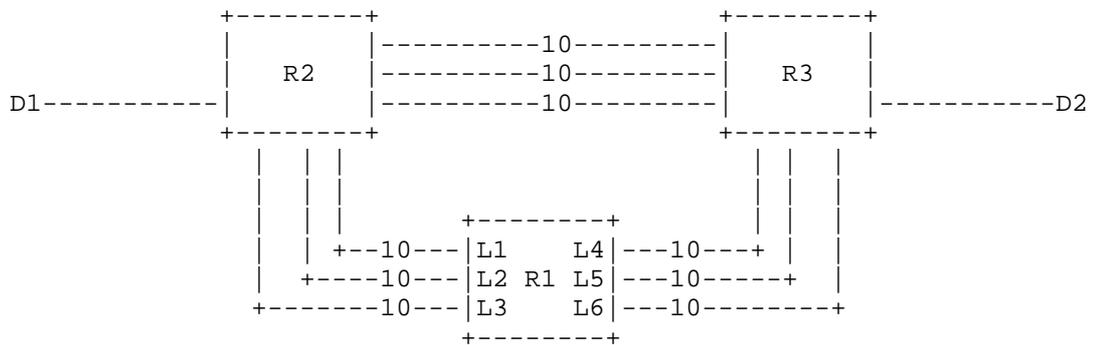


Figure 1: Example Topology

Consider the topology as show in Figure 1. The device R1 uses a links L1, L2 and L3 to carry traffic to destination D1. Similarly it uses links L4, L5 and L6 for destination D2. However in the event of links L1 and/or L2 fails traffic is still forwarded on link L3 causing traffic congestion.

In such situations operators will prefer the traffic for destination D1 is forwarded on L4, L5 and L6, as there is lesser chance of congestion. Similarly when links L4 and L5 also fails, the operator will prefer the traffic for D1 is forwarded is switched back on link L3 again. This document proposes a method called Bandwidth Based Metrics (hereafter referred as BBM), which helps achieving this desired behaviour.

BBM, on detecting a local link event, attempts to re-route traffic, based on remaining bandwidth across the links on the primary and alternate paths. When the remaining available bandwidth on the primary link(s) goes below a permissible limit (to be specified by the operator), traffic should be re-routed to one or more groups of alternative paths, and re-distributed onto multiple alternate paths with lesser likeliness of congesting them.

This document also specifies how to extend Fast Re-Route (FRR) for BBM to meet stringent re-convergence time constraints, and minimize traffic loss due to network congestion caused by standard FRR mechanisms.

2. BBM Concepts

2.1. Interface-Group

BBM method proposed in this document requires grouping of all the local links (or interfaces) attached to a node into one or more logical bundles. Such a logical grouping of multiple local interfaces is called an interface-group, and needs to be provisioned manually by the operator on each node. While assigning the local interfaces to a interface-group, all links connecting the local node to the same one-hop neighbor, SHOULD be assigned to a single interface-group. In other words the number of interface-group to be created on a node SHOULD be at the least, the number of one-hop neighbor nodes the particular node is connected to.

In Figure 1 links L1, L2, and L3 connecting R1 and R2 can be grouped into a single interface group (say IG1) on both R1 and R2. Similarly links L4, L5 and L6 connecting R1 and R3 can be grouped into another single interface-group (say IG2) on both R1 and R3.

2.2. BBM Metric Configurations

All the interfaces under a given interface-group shall share a metric that is proportionate to the cumulative bandwidth available using the individual links under the interface-group. if a link is associated with interface-group then interface-group metric MUST override individual link metric configuration. Implementations SHOULD allow operator to specify what metric should be associated for a given total remaining available bandwidth for each interface group. Implementations SHOULD also allow operator specify the default metric to be used for each interface-group.

In Figure 1, considering all the links L1 to L6 having bandwidth capacity of 100G each, and assigned into two interface-groups IG1 and IG2 (as shown in Section 2.1), following is an example of simple BBM config for each of these interface-group.

```
IG1:
  Member-Links: L1,L2,L3
  Total-Available-BW: 200G, Metric: 10
  Total-Available-BW: 100G, Metric: 50
  Default-Metric: 1000

IG2:
  Member-Links: L4,L5,L6
  Total-Available-BW: 200G, Metric: 10
  Total-Available-BW: 100G, Metric: 50
  Default-Metric: 1000
```

Figure 2: Example BBM Configuration

2.3. BBM Terminologies

This document also defines the following attributes to be associated with each interface-group.

Attribute	Value and Significance
Intf_List	The list of interfaces assigned to this group as per configuration.
BW_Curr	Total available bandwidth across all active member interfaces of this group.
BBM_Metric_Cfg_List	This is an array of "BBM_Metric_Cfg_Entry" (defined below). The key to the list is "bandwidth" and is always sorted in descending order (i.e. entries with higher "bandwidth" appears before entries with lower "bandwidth").
BBM_Metric_Cfg_Entry	This defines a single entry in "BBM_Metric_Cfg_List" array (defined above). It is a tuple ["Bandwidth", "Metric"], and defines the metric that should be associated with the individual interfaces of this group, when the total available bandwidth for the group matches "bandwidth" range specified in this entry. Refer to Table 2 for more details.
Default_Metric	The default metric as per configuration. Default metric will be assigned to all interfaces under this group if total available bandwidth for the group Does not match the "Bandwidth" range specified in any "BBM_Metric_Cfg_Entry" for this group. Refer to Table 2 for more details.

Table 1: Interface Group Attributes

2.4. Metric Derivation

Once a interface has been assigned to a interface-group, and the corresponding BBM metric configurations has been provisioned, metric to be associated with the member interfaces can be derived as follows:

```

Sort igp.BBM_Metric_Cfg_List in descending order based on BBM_Metric_Cfg_Entry.B
andwidth
Set Intf.Metric = 0
For (all BBM_Metric_Cfg_Entry in igp.BBM_Metric_Cfg_List
    in descending order)
  - If (igp.BW_Curr >=
        igp.BBM_Metric_Cfg_List.BBM_Metric_Cfg_Entry.Bandwidth)
    - Set Intf.Metric =
        igp.BBM_Metric_Cfg_List.BBM_Metric_Cfg_Entry.Metric.
      end the loop.
If (Intf.Metric == 0)
  - Set Intf.Metric = igp.Default_Metric.

```

Considering the BBM metric configurations for interface-group IG1 in Figure 2, Table 2 below shows how metric for individual interfaces of IG1 SHALL be computed at any point of time.

Active-Links	Total-Available-BW	BBM-Metric	Remarks
L1, L2, L3	300G	10	Total-Available-BW >= 200
L1, L2, L3(down)	200G	10	Total-Available-BW >= 200
L1, L2(down), L3(down)	100G	50	200 > Total-Available-BW >= 100

Table 2: BBM Metric Calculation

3. Bandwidth Based Routing

Once the metric of individual interfaces are derived from the corresponding interface-group BBM configuration, the same are used in the local IGP SPF computations. In addition to using the metrics in SPF computations, the same are also advertised as the corresponding link cost (instead of the original cost associated with the individual links) in the locally-generated IGP link-state advertisements. This is done to eliminate any looping possible otherwise.

Considering the topology in Figure 1, Table 3 below shows how traffic for destination D1 shall be re-routed based on a series of events and BBM metric configurations as shown in Figure 2.

Event	Interface-Group	Active-Links/Total-Available-BW	Total-Metric	Shortest Path
Initially	IG1	{L1, L2, L3} / 300G	10 + Dopt (R2,D1)	YES
	IG2	{L4, L5, L6} / 300G	20 + Dopt (R2,D1)	NO
L1 goes down	IG1	{L2, L3} / 200G	10 + Dopt (R2,D1)	YES
	IG2	{L4, L5, L6} / 300G	20 + Dopt (R2,D1)	NO
L2 goes down	IG1	{L2, L3} / 200G	50 + Dopt (R2,D1)	NO
	IG2	{L5, L6} / 200G	20 + Dopt (R2,D1)	YES
L4 goes down	IG1	{L3} / 100G	50 + Dopt (R2,D1)	NO
	IG2	{L5, L6} / 200G	20 + Dopt (R2,D1)	YES
L5 goes down	IG1	{L3} / 100G	50 + Dopt (R2,D1)	YES
	IG2	{L6} / 100G	60 + Dopt (R2,D1)	NO

Table 3: BBM based Routing

4. Bandwidth-based Fast Reroute

4.1. Overview

The BBM solution described in Section 2 requires IGPs running on the control plane of the network device, to detect the link failures, determine remaining available bandwidth, re-compute new optimum paths, and finally install the new best paths to the forwarding plane. This may take some time (in the order of 500 ms) for the traffic to switch to a better path.

Also, even if regular FRR mechanism using LFA [RFC5286] and Remote-LFA [I-D.ietf-rtgwg-remote-lfa] has been deployed, the alternate paths chosen is not guaranteed to meet bandwidth constraints. Also, though, [RFC5286] does not specify anything, most LFA implementations in link-state protocols running on the network devices around the world, employs use of a single backup link. Also if there are multiple primary interfaces for a specific destinations, most implementations do not install a alternate path in the forwarding plane. So in the event of the primary link (or one of the multiple primary links) going down, traffic is either switched to a single interface, or not switched to any other link at all. In the first case, there is more likeliness of the single alternate path getting congested (as it might be already carrying some primary traffic for other destinations already). In the latter case, there is more likeliness of causing a congestion on the remaining primary links (e.g. for destination D1, if both L1 and L2 goes down R1 still keeps the traffic on L3 during local repair, trying to push 300G traffic on a single 100G link L5).

Service providers who have stringent bandwidth requirements would need the device to switch the traffic during local repair to multiple alternate paths that have bandwidth constraints satisfied. When the remaining primary OR alternate paths alone cannot satisfy bandwidth requirements, it will also be desirable, to redistribute the traffic over a combination of primary AND alternate paths, during local repair as well as next SPF computations in IGP.

This document proposes a solution the above problem, based on combination of BBM logic (referred to in Section 2) and protection using LFA [RFC5286] and Remote-LFA [I-D.ietf-rtgwg-remote-lfa]. It requires a group of primary links to be protected using multiple non-best feasible alternate paths. The same group of alternate links shall also be pre-installed in forwarding table to facilitate fast re-route (FRR). The details of the solution is specified in the following sub-sections.

4.2. Assumptions and Pre-requisites

Following are some of the assumptions that the solution proposed in this document is based on.

The forwarding plane SHOULD be able handle multiple paths per route and let control plane set the preference for each path over the others. The forwarding machinery shall utilize this, to select a subset of preferred paths, and use them to forward actual traffic at any given point in time. Forwarding machinery SHOULD also load balance traffic with next-hops having same preference

All the links attached to the network device are bundled to create one or more interface-group(s). Also a link MUST belong to one and only one interface-group.

Loops are possible if protection is enabled on all three routes R1, R2 and R3 as shown in Figure 1. To avoid loops implementation MUST have downstream Path Criterion as explained in LFA [RFC5286]

For each interface-group, operator MAY enable protection by configuring the following two parameters.

Minimum-bandwidth: When the remaining bandwidth goes below this the outgoing traffic can no more be carried entirely on this bundle. Some of it shall be distributed across links of other best/non-best interface-groups.

Restore-bandwidth: When the remaining bandwidth exceeds this, the outgoing traffic can entirely be back over the members of this bundle and there is no need to use any other backup for all destinations reachable over the links of the bundle.

4.3. Additional Configuration and Attributes

This document defines the following configuration parameters to be associated with each interface-group for facilitating Bandwidth-based Fast Re-Route. Implementations MUST allow operators to configure these parameters for each interface-group on a network device that implements this solution.

Attribute	Value and Significance
Min_BW	This is the minimum bandwidth below which outgoing traffic MUST not be carried on this interface-group. It needs to load-balance across links of best/non-best interface-groups as well.
Restore_BW	This is the bandwidth above which the outgoing traffic MUST entirely be carried over the members of this interface-group not needing to load-balance across member links of other non-best interface-groups, provided it provides a path with shortest metric.

Table 4: BBM FRR Configurations

In Figure 1, considering all the links L1 to L6 having bandwidth capacity of 100G each, and assigned into two interface-groups IG1 and IG2 (as shown in Section 2.1), following is an example of simple BBM FRR config for each of these interface-group.

```

IG1:
  Member-Links: L1,L2,L3
  Total-Available-BW: 200G, Metric: 10
  Total-Available-BW: 100G, Metric: 50
  Default-Metric: 1000
  Protection: Enabled
    Restore-Bandwidth: 200G
    Min-Bandwidth: 100G

IG2:
  Member-Links: L4,L5,L6
  Total-Available-BW: 200G, Metric: 10
  Total-Available-BW: 100G, Metric: 50
  Default-Metric: 1000
  Protection: Enabled
    Restore-Bandwidth: 200G
    Min-Bandwidth: 100G

```

Figure 3: Example BBM FRR Configuration

This document defines the following attributes to be associated with each interface-group for facilitating Bandwidth-based Fast Re-Route.

Attribute	Value and Significance
BW_PostFail	Cumulative bandwidth through all the remaining primary next-hops considering the primary next-hop with highest bandwidth goes down.
Metric_PostFail	Bandwidth based metric after a link goes down.

Table 5: Additional Interface-Group Attributes

This solution proposed in this document also requires IGPs to define and associated the following attributes for each destination node in the IGP link-state database.

Attribute	Value and Significance
Pri_Nh_Count	Number of primary next-hops found for the destination.
Pri_BW_Curr	Cumulative bandwidth across all the remaining primary next-hops.
Pri_BW_PostFail	Cumulative bandwidth through all the remaining primary nexthops considering the primary nexthop with highest bandwidth goes down.
Pri_BW_Restore	Cumulative restore-bandwidth for all the interface-groups considered for primary nexthops.
Pri_BW_Min	Cumulative minimum-bandwidth for all the interface-groups considered for primary nexthops.

Table 6: Per-node Attributes

4.4. Enhancements to Local Repair in Forwarding Plane

Additionally, the solution proposed in this document also mandates, that the forwarding plane SHOULD implement the following enhanced local-repair logic, to facilitate BBM based fast-re-route, on detecting a link-down event.

For each affected prefix (a prefix is affected if the fated link was one of the preferred active paths used for forwarding).

- Find the actual affected path, and mark it unusable.
- For all other paths downloaded from control-plane,
 - If the preference is same as that of the affected path,
 - Modify its preference to value even lower than normal backup paths.

Finally, go through all remaining active paths

- Select a subset of paths (that share the same highest preference among all),
- Use the selected subset of paths to actually forward traffic.

Figure 4: Enhanced Local Repair in Forwarding Plane

4.5. Influencing Path Preferences

Like mentioned in Section 4.2 the solution proposed in this document relies on the preference-based local-repair logic implemented in forwarding-plane to facilitate fast re-route. This solution requires IGPs to indirectly influence the local-repair action taken by the forwarding-plane by choosing an suitable alternate path with an appropriate preference-value pre-computed and installed in the forwarding-plane, well ahead of the actual link failure event.

Table 7 below, specifies a set path-preference types that this document proposes IGP to define and use while downloading any path for a given destination in the forwarding table.

Preference-Type	Significance
Pri_Nh_Pref	Preference type for normal primary paths.
Bkup_Nh_Pref_High	Preference type for paths, which are preferred, more than normal backup paths but less compared to normal primary paths.
Bkup_Nh_Pref_Normal	Preference type for normal backup paths.

Table 7: Path-Preference Types

4.6. Path Selection and Preference

Based on the above assumptions, additional configuration parameters and attributes the document proposes IGPs to implement the following logic for computing primary and alternate paths for each destination, and determine their corresponding path-preference value as well

Step 1:

=====

- For each interface-group "igp"
 - Update "igp.BW_Curr" by adding the bandwidths of the individual active member interfaces.
 - Update "igp.BW_PostFail", assuming one of the active member interfaces with highest bandwidth goes down next.

Step 2:

=====

- For each destination node "D" in the network (Pass-1)
 - Update D.Pri_BW_Restore and D.Pri_BW_Min from the SPF results.
 - Reset D.Pri_Nh_Count to 0.
 - For each corresponding primary path N,
 - Set "igp" -> Interface-group N belongs to.
 - If igp.BW_Curr > D.Min_BW
 - Set preference of N to Pri_Nh_Pref.
 - Increment the D.Pri_Nh_Count by 1.
 - Else
 - Set preference of N to Bkup_Nh_Pref_Normal.

Step 3:

=====

- For each destination node "D" in the network (Pass-2)
 - Update D.Pri_BW_Restore and D.Pri_BW_Min.
 - For each corresponding primary path N,
 - Set "Pri_Igp" -> Interface-group N belongs to.
 - If protection configured on "Pri_Igp"
 - If igp.Pri_BW_PostFail < D.Pri_BW_Restore,
OR igp.Pri_BW_PostFail <= D.Pri_BW_Min
 - For all backup paths M,
 - Set "Alt_Igp" -> Interface-group N belongs to.
 - Select M for installing in forwarding plane.
 - If D.Pri_Nh_Count == 0
 - If Alt_igp.BW_Curr >= D.Pri_BW_Restore,
AND Alt_Igp.BW_Curr > D.Pri_BW_Min
 - Set preference of M to Pri_Nh_Pref.
 - Else
 - Set preference of M to Bkup_Nh_Pref_Normal.
 - Else
 - If Alt_Igp.BW_Curr >= D.Pri_BW_Restore,
AND Alt_Igp.BW_Curr > D.Pri_BW_Min
 - Set preference of M to Bkup_Nh_Pref_High.
 - Else
 - Set preference of M to Bkup_Nh_Pref_Normal.

5. Limitations

The BBM method proposed in this document does NOT ensure end to end bandwidth requirement. It, only ensures that the metric is altered only on local interfaces, based on the BBM metric configurations and remaining available bandwidth.

The solution proposed in this documents attempts to provide protection for single link failures only. It always assumes that link with the highest individual bandwidth capacity shall fail next. In case if any other link with lesser individual bandwidth capacity fails instead, the local repair action taken by the forwarding plane may not be exactly as expected, even though the forwarding plane will still take care of protecting the traffic.

6. Security Consideration

Changes suggested in the draft does not raise any security concerns.

7. IANA Consideration

This draft does not have any request from IANA.

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>>.

8.2. Informative References

[I-D.ietf-rtgwg-remote-lfa] Bryant, S., Filtsils, C., Previdi, S., Shand, M., and N. So, "Remote Loop-Free Alternate (LFA) Fast Re-Route (FRR)", draft-ietf-rtgwg-remote-lfa-11 (work in progress), January 2015.

[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>>.

Authors' Addresses

Santosh Pallagatti (editor)
Juniper Networks
Embassy Business Park
Bangalore, KA 560093
India

Email: santoshpk@juniper.net

Pushpasis Sarkar (editor)
Juniper Networks
Embassy Business Park
Bangalore, KA 560093
India

Email: psarkar@juniper.net

Hannes Gredler
Juniper Networks
1194 N. Mathilda Ave.
Sunnyvale, California 94089-1206
USA

Email: hannes@juniper.net

Stephane Litkowski
Orange Business Service

Email: stephane.litkowski@orange.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 7, 2015

N. Wu
S. Zhuang
Huawei
March 6, 2015

A YANG Data Model for Flow Specification
draft-wu-rtgwg-flowspec-cfg-00

Abstract

This document defines a YANG data model for Flow Specification implementations. 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 September 7, 2015.

Copyright Notice

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

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

to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction 2
 1.1. Terminology 2
 1.2. Tree Diagrams 3
 2. Flow Specification data model 3
 3. Flow Specification YANG Module 7
 4. IANA Considerations 20
 5. Security Considerations 20
 6. Acknowledgments 20
 7. References 21
 7.1. Normative References 21
 7.2. Informative References 21
 Authors' Addresses 21

1. Introduction

This document defines a YANG [RFC6020] data model for Flow Specification [RFC5575] implementations.

The data model covers configuration of filtering rules and actions of traffic flow specification. It also provides information about operational running state of traffic flow specification, which includes RIB of flow specification route, attributes dedicated to different protocols and statistics.

1.1. Terminology

The following terms are defined in [RFC6020]:

- o configuration data
- o data model
- o module
- o 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:

- o Brackets "[" and "]" enclose list keys.
- o Abbreviations before data node names: "rw" means configuration data (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. Flow Specification data model

The data model has the following structure for configuration of traffic flow specification:

```

+--rw flowspec-cfg
|
|  +--rw flow-route* [name]
|  |
|  |  +--rw name                string
|  |  +--rw filter-rules
|  |  |
|  |  |  +--rw destination-prefix
|  |  |  |
|  |  |  |  +--rw dest-prefix-length?  uint8
|  |  |  |  +--rw dest-prefix?        inet:ip-address
|  |  |  +--rw source-prefix
|  |  |  |
|  |  |  |  +--rw src-prefix-length?   uint8
|  |  |  |  +--rw src-prefix?         inet:ip-address
|  |  |  +--rw ip-protocol
|  |  |  |
|  |  |  |  +--rw op-protocol* [compare-protocol ip-PROTO]
|  |  |  |  |
|  |  |  |  |  +--rw compare-protocol  compare-ops
|  |  |  |  |  +--rw ip-PROTO        ip-protocol-types
|  |  |  +--rw port-number
|  |  |  |
|  |  |  |  +--rw op-port* [compare-port port]
|  |  |  |  |
|  |  |  |  |  +--rw compare-port    compare-ops
|  |  |  |  |  +--rw port            uint16
|  |  |  +--rw destination-port
|  |  |  |
|  |  |  |  +--rw op-destport* [compare-destport dest-port]
|  |  |  |  |
|  |  |  |  |  +--rw compare-destport  compare-ops
|  |  |  |  |  +--rw dest-port        uint16
|  |  |  +--rw source-port

```



```

+--ro index                               uint32
+--ro islocal                             boolean
+--ro local-name?                         string
+--ro neighbor?                          inet:ip-address
+--ro duration?                           uint32
+--ro filter-rules
  |-- destination-prefix
  |   |-- dest-prefix-length?             uint8
  |   |-- dest-prefix?                   inet:ip-address
  |-- source-prefix
  |   |-- src-prefix-length?             uint8
  |   |-- src-prefix?                   inet:ip-address
  |-- ip-protocol
  |   |-- op-protocol* [compare-protocol ip-protol]
  |   |   |-- compare-protocol           compare-ops
  |   |   |-- ip-protol                 ip-protocol-types
  |-- port-number
  |   |-- op-port* [compare-port port]
  |   |   |-- compare-port             compare-ops
  |   |   |-- port                     uint16
  |-- destination-port
  |   |-- op-destport* [compare-destport dest-port]
  |   |   |-- compare-destport         compare-ops
  |   |   |-- dest-port                uint16
  |-- source-port
  |   |-- op-srcport* [compare-srcport src-port]
  |   |   |-- compare-srcport          compare-ops
  |   |   |-- src-port                 uint16
  |-- icmp-type-value
  |   |-- op-icmp-type* [compare-icmp-type icmp-type]
  |   |   |-- compare-icmp-type        compare-ops
  |   |   |-- icmp-type                uint8
  |-- icmp-code-value
  |   |-- op-code-value* [compare-icmp-code icmp-code]
  |   |   |-- compare-icmp-code        compare-ops
  |   |   |-- icmp-code                uint8
  |-- tcp-flag
  |   |-- op-bitmask* [tcp-flag-op tcp-flag-value]
  |   |   |-- tcp-flag-op              bitmask-ops
  |   |   |-- tcp-flag-value           uint16
  |-- pkt-len
  |   |-- op-pkt-len* [compare-pkt-len packet-length]
  |   |   |-- compare-pkt-len          compare-ops
  |   |   |-- packet-length            uint16
  |-- dscp
  |   |-- op-dscp* [compare-dscp dscp-value]
  |   |   |-- compare-dscp             compare-ops
  |   |   |-- dscp-value               dscp-type

```

```

|      |--ro fragment
|      |   |--ro op-fragment* [compare-fragment fragment-mode]
|      |   |   |--ro compare-fragment    bitmask-ops
|      |   |   |--ro fragment-mode      fragment-mode-type
|--ro filter-actions
|   |--ro traffic-rate
|   |   |--ro rate?    float
|--ro redirect
|   |--ro route-target?    string
|--ro traffic-marking
|   |--ro remark-dscp?    union
|--ro traffic-deny
|   |--ro deny?    boolean
|--ro protocol-specific
|   |--ro (protocol)?
|   |   |--:(bgp)
|   |   |   |--ro peer-rid?        inet:ipv4-address
|   |   |   |--ro isrefclient?    boolean
|   |   |   |--ro med?            uint32
|   |   |   |--ro preference?     uint32
|   |   |   |--ro local-pref?     uint8
|   |   |   |--ro origin?         enumeration
|   |   |   |--ro originator?     inet:ipv4-address
|   |   |   |--ro community?     string
|   |   |   |--ro ext-community?  string
|   |   |   |--ro cluster-list?  string
|   |   |   |--ro path-as?       string
|   |   |--:(ospf)
|   |   |   |--ro lsa-type?       string
|   |   |   |--ro ls-id?         inet:ipv4-address
|   |   |   |--ro advertiser?    inet:ipv4-address
|   |   |   |--ro area?         union
|   |   |   |--ro tag?          uint32
|   |   |--:(isis)
|   |   |   |--ro lsp-id?        string
|   |   |   |--ro priority?     uint32
|   |   |   |--ro admin-tag?    uint32
|--ro flowspec-statis

```

This data model defines two state containers for traffic flow specification. In the first container, there is a list of state containers per flow specification route, which contains the current state for filtering rules and actions. In the second container, there is statistics information for traffic flow specifications.

3. Flow Specification YANG Module

```
//<CODE BEGINS> file "ietf-flowspec@2015-02-27.yang"
```

```
module ietf-flowspec {
  namespace "urn:huawei:params:xml:ns:yang:flowspec";
  prefix flowspec;

  import ietf-inet-types {
    prefix inet;
  }

  import ietf-yang-types {
    prefix yang;
  }

  organization
    "IETF RTGWG (Routing Working Group) Working Group";

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

    WG Chair: Chris Bowers
              <mailto:cbowers@juniper.net>

    WG Chair: Jeff Tantsura
              <mailto:jeff.tantsura@ericsson.com>

    Editor: Shunwan Zhuang
            <mailto:zhuangshunwan@huawei.com>

    Editor: Nan Wu
            <mailto:eric.wu@huawei.com>";
```

description

"This module contains a collection of YANG definitions for configuring flow specification implementations.

Copyright (c) 2014 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>)."

```
revision 2015-02-27 {
  description
    "Initial revision.";
  reference
    "draft-ietf-netmod-routing-cfg-16:
    A YANG Data Model for Routing Management";
}

/* Typedefs */
typedef compare-ops {
  type enumeration {
    enum greater-than;
    enum less-than;
    enum equal;
  }
}

typedef bitmask-ops {
  type enumeration {
    enum match;
    enum not;
  }
}

typedef ip-protocol-types {
  type enumeration {
    enum icmp {
      value 1;
    }
    enum igmp {
      value 2;
    }
    enum ipip {
      value 4;
    }
    enum tcp {
      value 6;
    }
    enum egp {
      value 8;
    }
    enum udp {
      value 17;
    }
    enum pv6 {
```

```
        value 41;
    }
    enum rsvp {
        value 46;
    }
    enum gre {
        value 47;
    }
    enum esp {
        value 50;
    }
    enum ospf {
        value 89;
    }
    enum pim {
        value 103;
    }
}

typedef dscp-remarked {
    type enumeration {
        enum be;
        enum cs1;
        enum af11;
        enum af12;
        enum af13;
        enum cs2;
        enum af21;
        enum af22;
        enum af23;
        enum cs3;
        enum af31;
        enum af32;
        enum af33;
        enum cs4;
        enum af41;
        enum af42;
        enum af43;
        enum cs5;
        enum ef;
        enum cs6;
        enum cs7;
    }
}

typedef dscp-type {
    type union {
```

```
    type dscp-remarked;
    type uint8;
  }
}

typedef fragment-mode-type {
  type enumeration {
    enum fragment {
      description
        "Indicates that fragments are checked.";
    }
    enum non-fragment {
      description
        "Indicates that non-fragmented packets are checked.";
    }
    enum fragment-spe-first {
      description
        "Indicates that the first fragmented packet is checked.";
    }
  }
}

/*Definition from
  http://www.ietf.org/mail-archive/web/netmod/current/msg02685.html
*/
typedef float {
  type union {
    type decimal64 {
      fraction-digits 14;
      range "-9999.9999999999999999 .. 9999.9999999999999999";
    }
    type decimal64 {
      fraction-digits 13;
      range "-99999.9999999999999999 .. 99999.9999999999999999";
    }
    type decimal64 {
      fraction-digits 12;
      range "-999999.9999999999999999 .. 999999.9999999999999999";
    }
    type decimal64 {
      fraction-digits 11;
      range "-9999999.9999999999999999 .. 9999999.9999999999999999";
    }
    type decimal64 {
      fraction-digits 10;
      range "-99999999.9999999999999999 .. 99999999.9999999999999999";
    }
    type decimal64 {
```

```

    fraction-digits 9;
    range "-999999999.999999999 .. 999999999.999999999";
  }
  type decimal64 {
    fraction-digits 8;
    range "-999999999.99999999 .. 999999999.99999999";
  }
  type decimal64 {
    fraction-digits 7;
    range "-9999999999.9999999 .. 9999999999.9999999";
  }
  type decimal64 {
    fraction-digits 6;
    range "-99999999999.999999 .. 99999999999.999999";
  }
  type decimal64 {
    fraction-digits 5;
    range "-999999999999.99999 .. 999999999999.99999";
  }
  type decimal64 {
    fraction-digits 4;
    range "-9999999999999.9999 .. 9999999999999.9999";
  }
  type decimal64 {
    fraction-digits 3;
    range "-99999999999999.999 .. 99999999999999.999";
  }
  type decimal64 {
    fraction-digits 2;
    range "-999999999999999.99 .. 999999999999999.99";
  }
  type decimal64 {
    fraction-digits 1;
    range "-9999999999999999.9 .. 9999999999999999.9";
  }
}
}

/* Grouping definition */
grouping flow-filter-rule {
  container destination-prefix {
    leaf dest-prefix-length {
      type uint8;
      description
        "Specifies the mask length.";
    }
    leaf dest-prefix {
      type inet:ip-address;
    }
  }
}

```

```
        description
          "Specifies the destination address of the traffic.";
      }
  }
  container source-prefix {
    leaf src-prefix-length {
      type uint8;
      description
        "Specifies the mask length.";
    }
    leaf src-prefix {
      type inet:ip-address;
      description
        "Specifies the source IP address of the traffic.";
    }
  }
  container ip-protocol {
    list op-protocol {
      key "compare-protocol ip-proto";
      leaf compare-protocol {
        type compare-ops;
        description
          "Indicates the comparing relationship between the destination
           port number and the specified one.";
      }
      leaf ip-proto {
        type ip-protocol-types;
        description
          "Specifies a protocol type with a name or number.";
      }
    }
  }
  container port-number {
    list op-port {
      key "compare-port port";
      leaf compare-port {
        type compare-ops;
        description
          "Specifies the operation symbol of comparing source
           or destination port numbers.";
      }
      leaf port {
        type uint16;
        description
          "Specifies the port number.";
      }
    }
  }
}
```

```
container destination-port {
  list op-destport {
    key "compare-destport dest-port";
    leaf compare-destport {
      type compare-ops;
      description
        "Specifies the operation symbol of comparing
         destination port numbers with designated value.";
    }
    leaf dest-port {
      type uint16;
      description
        "Specifies the destination port number of the traffic.";
    }
  }
}
container source-port {
  list op-srcport {
    key "compare-srcport src-port";
    leaf compare-srcport {
      type compare-ops;
      description
        "Specifies the operation symbol of comparing
         source port numbers with designated value.";
    }
    leaf src-port {
      type uint16;
      description
        "Specifies the source port number of the traffic.";
    }
  }
}
container icmp-type-value {
  list op-icmp-type {
    key "compare-icmp-type icmp-type";
    leaf compare-icmp-type {
      type compare-ops;
      description
        "Specifies the operation symbol of comparing
         icmp type value with designated value.";
    }
    leaf icmp-type {
      type uint8;
      description
        "Specifies the ICMP packet type.";
    }
  }
}
```

```
container icmp-code-value {
  list op-code-value {
    key "compare-icmp-code icmp-code";
    leaf compare-icmp-code {
      type compare-ops;
      description
        "Specifies the operation symbol of comparing
        icmp code value with designated value.";
    }
    leaf icmp-code {
      type uint8;
      description
        "Specifies the ICMP packet code.";
    }
  }
}
container tcp-flag {
  list op-bitmask {
    key "tcp-flag-op tcp-flag-value";
    leaf tcp-flag-op {
      type bitmask-ops;
      description
        "Indicates that traffic matching the TCP flag value
        can or can not pass the filtering rule.";
    }
    leaf tcp-flag-value {
      type uint16;
      description
        "Specifies the TCP flag value.";
    }
  }
}
container pkt-len {
  list op-pkt-len {
    key "compare-pkt-len packet-length";
    leaf compare-pkt-len {
      type compare-ops;
      description
        "Specifies the operation symbol of comparing
        packet length with designated value.";
    }
    leaf packet-length {
      type uint16;
      description
        "Specifies IP packet length.";
    }
  }
}
```

```
    container dscp {
      list op-dscp {
        key "compare-dscp dscp-value";
        leaf compare-dscp {
          type compare-ops;
          description
            "Specifies the operation symbol of comparing
             dscp value with designated value.";
        }
        leaf dscp-value {
          type dscp-type;
          description
            "Specifies the DSCP value.";
        }
      }
    }
  }
}

container fragment {
  list op-fragment {
    key "compare-fragment fragment-mode";
    leaf compare-fragment {
      type bitmask-ops;
      description
        "Indicates that traffic matching the fragment type
         or not can pass the filtering rule.";
    }
    leaf fragment-mode {
      type fragment-mode-type;
      description
        "Indicates the fragment mode that is checked.";
    }
  }
}
}

grouping flow-filter-action {
  container traffic-rate {
    leaf rate {
      type float;
      description
        "Specifies the traffic rate in IEEE floating point
         [IEEE.754.1985] format, units being bytes per second.";
    }
  }
  container redirect {
    leaf route-target {
      type string {
        length "3..21";
      }
    }
  }
}
```

```
        description
          "Specifies the name of a target VPN to which
           attack traffic is redirected.";
      }
    }
  container traffic-marking {
    leaf remark-dscp {
      type union {
        type dscp-remarked;
        type uint8 {
          range "0..63";
        }
      }
      description
        "Specifies a re-marked DSCP value.";
    }
  }
  container traffic-deny {
    leaf deny {
      type boolean;
      description
        "Enables a device to discard the traffic matching
         a filtering rule.";
    }
  }
}

grouping flow-proto-specific {
  choice protocol {
    case bgp {
      leaf peer-rid {
        type inet:ipv4-address;
      }
      leaf isrefclient {
        type boolean;
      }
      leaf med {
        type uint32;
      }
      leaf preference {
        type uint32;
      }
      leaf local-pref {
        type uint8;
      }
      leaf origin {
        type enumeration {
          enum igp {

```

```
        value "0";
      }
      enum egg {
        value "1";
      }
      enum incomplete {
        value "2";
      }
    }
  }
  leaf originator {
    type inet:ipv4-address;
  }
  leaf community {
    type string;
  }
  leaf ext-community {
    type string;
  }
  leaf cluster-list {
    type string;
  }
  leaf path-as {
    type string;
  }
}
case ospf {
  leaf lsa-type {
    type string;
  }
  leaf ls-id {
    type inet:ipv4-address;
  }
  leaf advertiser {
    type inet:ipv4-address;
  }
  leaf area {
    type union {
      type uint32;
      type inet:ipv4-address;
    }
  }
  leaf tag {
    type uint32;
  }
}
case isis {
  leaf lsp-id {
```

```
        type string;
    }
    leaf priority {
        type uint32;
    }
    leaf admin-tag {
        type uint32;
    }
}
}
}

/* Configuration data nodes */
container flowspec-cfg {
    description
        "Configuration for flow specification.";
    list flow-route {
        key "name";
        description
            "Configuration of a flow route list.";
        leaf name {
            description
                "The name of a flow route.";
            type string;
        }
        container filter-rules {
            description
                "Configuration for filter rules.";
            uses flow-filter-rule;
        }
        container filter-actions {
            description
                "Configuration for filter actions.";
            uses flow-filter-action;
        }
    }
}

/* Operational state data nodes */
container flowspec-state {
    config "false";
    description
        "Operational state of flow specification.";

    container flowspec-rib {
        list flowspec-route {
            key "index islocal";
            leaf index {
```


4. IANA Considerations

This document registers a URI in the "IETF XML Registry" [RFC3688]. Following the format in RFC 3688, the following registration has been made.

URI: urn:ietf:params:xml:ns:yang:ietf-flowspec

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-flowspec

Namespace: urn:ietf:params:xml:ns:yang:ietf-flowspec

Prefix: flowspec

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. Acknowledgments

TBD.

7. References

7.1. Normative References

- [RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, January 2004.
- [RFC5575] Marques, P., Sheth, N., Raszuk, R., Greene, B., Mauch, J., and D. McPherson, "Dissemination of Flow Specification Rules", RFC 5575, August 2009.
- [RFC6020] Bjorklund, M., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, October 2010.
- [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, June 2011.
- [RFC6536] Bierman, A. and M. Bjorklund, "Network Configuration Protocol (NETCONF) Access Control Model", RFC 6536, March 2012.

7.2. Informative References

- [I-D.liang-ospf-flowspec-extensions]
Liang, Q., You, J., and N. Wu, "OSPF Extensions for Flow Specification", draft-liang-ospf-flowspec-extensions-02 (work in progress), November 2014.
- [I-D.you-isis-flowspec-extensions]
You, J. and Q. Liang, "IS-IS Extensions for Flow Specification", draft-you-isis-flowspec-extensions-00 (work in progress), September 2014.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

Authors' Addresses

Nan Wu
Huawei
Huawei Bld., No.156 Beiqing Rd.
Beijing 100095
China

Email: eric.wu@huawei.com

Shunwan Zhuang
Huawei
Huawei Bld., No.156 Beiqing Rd.
Beijing 100095
China

Email: zhuangshunwan@huawei.com

Network Working Group
Internet-Draft
Intended status: Informational
Expires: June 30, 2015

G. Yan
S. Zhuang
Huawei Technologies
December 27, 2014

Yang Data Model for Routing Policy
draft-yan-rtgwg-routing-policy-yang-00

Abstract

This document defines a YANG data model that can be used to configure and manage routing policies.

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 30, 2015.

Copyright Notice

Copyright (c) 2014 IETF Trust and the persons identified as the document authors. All rights reserved.

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

include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
2. Definitions and Acronyms	2
3. Introduction to Routing Policies	3
4. Design of Data Model	4
4.1. Overview	4
4.2. AS_Path Filter Configuration	5
4.3. Community Filter Configuration	6
4.4. Extend Community Filter Configuration	6
4.5. Extend Community SoO Lists Configuration	7
4.6. RD Filters Configuration	7
4.7. IPv4 Prefix Filters Configuration	8
4.8. IPv6 Prefix Filters Configuration	8
4.9. Route-Policy Configuration	9
5. Route Policy Yang Module	11
6. IANA Considerations	50
7. Security Considerations	50
8. Acknowledgements	50
9. References	50
Authors' Addresses	50

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 programatic APIs.

This document defines a YANG data model that can be used to configure and manage routing policies.

2. Definitions and Acronyms

- ACL: Access Control List
- AS: Autonomous System
- BGP: Border Gateway Protocol

JSON: JavaScript Object Notation

NETCONF: Network Configuration Protocol

PBR: Policy-Based Routing

RD: Route Distinguisher

RPKI: Resource Public Key Infrastructure

VPN: Virtual Private Network

YANG: A data definition language for NETCONF

3. Introduction to Routing Policies

Routing policies are used to filter routes and set attributes for routes. Changing route attributes (including reachability) changes the path that network traffic passes through.

The difference between a routing policy and policy-based routing (PBR) is as follows:

- o Routing policies apply to routes. Based on routing protocols, the result of route generation, advertisement, and selection is changed by following rules, changing parameters, or using control modes. That is, the contents in the routing table are changed.
- o PBR applies to data packets. PBR provides a means to route or forward data packets flexibly based on predefined policies instead of following the routes in the existing routing table.

When advertising, receiving, and importing routes, the router implements certain policies based on actual networking requirements to filter routes and change the attributes of the routes. Routing policies serve the following purposes:

- o Control route advertising: Only routes that match the rules specified in a policy are advertised.
- o Control route receiving: Only the required and valid routes are received. This reduces the size of the routing table and improves network security.
- o Filter and control imported routes: A routing protocol may import routes discovered by other routing protocols. Only routes that satisfy certain conditions are imported to meet the requirements of the protocol.

- o Modify attributes of specified routes Attributes of the routes: that are filtered by a routing policy are modified to meet the requirements of the local device.

- o Configure fast reroute (FRR): If a backup next hop and a backup outbound interface are configured for the routes that match a routing policy, IP FRR, VPN FRR, and IP+VPN FRR can be implemented.

Routing policies are implemented using the following procedures:

- 1) Define rules: Define features of routes to which routing policies are applied. Users define a set of matching rules based on different attributes of routes, such as the destination address and the address of the router that advertises the routes.

- 2) Implement the rules: Apply the matching rules to routing policies for advertising, receiving, and importing routes.

4. Design of Data Model

4.1. Overview

The routing policy Yang module is divided in following containers :

- o asPathFilters : An AS_Path filter is used to filter BGP routes based on AS_Path attributes contained in the BGP routes..

- o communityFilters : A community filter is used to filter BGP routes based on the community attributes contained in the BGP routes.

- o extendCommunityFilters : An extcommunity filter is used to filter BGP routes based on extended community attributes.

- o extendCommunitySooLists : A SoO extended community filter is used to filter BGP routes based on SoO extended community attributes.

- o rdFilters : An RD filter is used to filter BGP routes based on RDs in VPN routes.

- o prefixFilters : An IP prefix list contains a group of route filtering rules.

- o ipv6PrefixFilters : An IPv6 prefix list contains a group of route filtering rules.

- o routePolicys : A Route-Policy is a complex filter. It is used to match attributes of specified routes and change route attributes when

specific conditions are met. A Route-Policy can use the preceding filters to define its matching rules.

The figure below describe the overall structure of the routing policy Yang module :

```

module: routing-policy
  +--rw routing-policy
    +--rw asPathFilters
    ...
    +--rw communityFilters
    ...
    +--rw extendCommunityFilters
    ...
    +--rw extendCommunitySooLists
    ...
    +--rw rdFilters
    ...
    +--rw prefixFilters
    ...
    +--rw ipv6PrefixFilters
    ...
    +--rw routePolicys
      +--rw routePolicy* [name]
        +--rw name string
        +--rw routePolicyNodes
          +--rw routePolicyNode* [nodeSequence]
            +--rw nodeSequence uint32
            +--rw matchMode? enumeration
            +--rw description? string
            +--ro matchCount? uint32
            +--rw matchCondition
              | +--rw matchCostValue uint32
              ...
              | +--rw matchExtCmntySooList? string
            +--rw applyAction
              +--rw applyAsPaths
              ...
              +--rw applyPriorityValue uint16
  
```

4.2. AS_Path Filter Configuration

An AS_Path filter uses the regular expression to define matching rules.

```

+--rw asPathFilters
|  +--rw asPathFilterKeyedByIndex* [index]
|  |  +--rw index                uint32
|  |  +--rw asPathFilterNodes
|  |  |  +--rw asPathFilterNode* [nodeSequence]
|  |  |  |  +--rw nodeSequence    uint32
|  |  |  |  +--rw matchMode?     enumeration
|  |  |  |  +--rw regular        string
|  |  +--rw asPathFilterKeyedByName* [name]
|  |  |  +--rw name                string
|  |  |  +--rw asPathFilterNodes
|  |  |  |  +--rw asPathFilterNode* [nodeSequence]
|  |  |  |  |  +--rw nodeSequence    uint32
|  |  |  |  |  +--rw matchMode?     enumeration
|  |  |  |  |  +--rw regular        string

```

4.3. Community Filter Configuration

A community filter is used to filter BGP routes based on the community attributes contained in the BGP routes. The community attribute is a set of destination addresses with the same characteristics. Therefore, filtering rules defined based on community attributes can be used to filter BGP routes..

```

+--rw communityFilters
|  +--rw basicCommunityFilterKeyedByIndex* [index]
|  |  +--rw index                uint32
|  |  +--rw basicCommunityNodes
|  |  |  +--rw basicCommunityNode* [nodeSequence]
|  |  ...
|  |  +--rw basicCommunityFilterKeyedByName* [name]
|  |  |  +--rw name                string
|  |  |  +--rw basicCommunityNodes
|  |  |  |  +--rw basicCommunityNode* [nodeSequence]
|  |  |  ...
|  |  +--rw advancedCommunityFilterKeyedByIndex* [index]
|  |  ...
|  |  +--rw advancedCommunityFilterKeyedByName* [name]
|  |  ...

```

4.4. Extend Community Filter Configuration

An extendCommunity filter is used to filter BGP routes based on extended community attributes. An extcommunity filter is used to filter only BGP routes because the extended community attribute is a private attribute of BGP.

```

+--rw extendCommunityFilters
|  +--rw basicExtendCommunityFilterKeyedByIndex* [index]
|  ..
|  +--rw basicExtendCommunityFilterKeyedByName* [name]
|  ...
|  +--rw advancedExtendCommunityFilterKeyedByIndex* [index]
|  ...
|  +--rw advancedExtendCommunityFilterKeyedByName* [name]
|  ...

```

4.5. Extend Community SoO Lists Configuration

An SoO extended community filter is used to filter BGP routes based on SoO extended community attributes.

```

+--rw extendCommunitySooLists
|  +--rw extendCommunitySooList* [name]
|  |  +--rw filterType          enumeration
|  |  +--rw name                string
|  |  +--rw basicExtendCommunitySooListNodes
|  |  ..
|  |  +--rw advanceExtendEommunitySooListNodes
|  |  |  +--rw advanceExtendEommunitySooListNode* [nodeSequence matchMode]
|  |  |  +--rw nodeSequence      uint32
|  |  |  +--rw matchMode         enumeration
|  |  |  +--rw regular           string
|  |  ..
|  ..

```

4.6. RD Filters Configuration

An RD filter is used to filter BGP routes based on RDs in VPN routes. RDs are used to distinguish IPv4 and IPv6 prefixes in the same address segment in VPN instances. RD filters specify matching rules regarding RD attributes.

```

+--rw rdFilters
|  +--rw rdFilter* [index]
|  |  +--rw index                uint32
|  |  +--rw rdFilterNodes
|  |  |  +--rw rdFilterNode* [nodeSequence]
|  |  |  +--rw nodeSequence      uint32
|  |  |  +--rw matchMode?       enumeration
|  |  |  +--rw rdStrings
|  |  |  |  +--rw rdString* [rdStringValue]
|  |  |  |  +--rw rdStringValue  string
|  |  ..
|  ..

```

4.7. IPv4 Prefix Filters Configuration

An IP prefix list contains a group of route filtering rules. Users can specify the prefix and mask length range to match the destination network segment address or the next hop address of a route. An IP prefix list is used to filter routes that are advertised and received by various dynamic routing protocols..

```
+--rw prefixFilters
|  +--rw prefixFilter* [name]
|  |   +--rw name                string
|  |   +--ro permitCnt?          uint32
|  |   +--ro denyCnt?            uint32
|  |   +--rw prefixFilterNodes
|  |   |   +--rw prefixFilterNode* [nodeSequence]
|  |   |   |   +--rw nodeSequence    uint32
|  |   |   |   +--rw matchMode?      enumeration
|  |   |   |   +--rw address          inet:ipv4-address
|  |   |   |   +--rw maskLength      uint8
|  |   |   |   +--rw matchNetwork?   boolean
|  |   |   |   +--rw greaterEqual?   uint8
|  |   |   |   +--rw lessEqual?      uint8
```

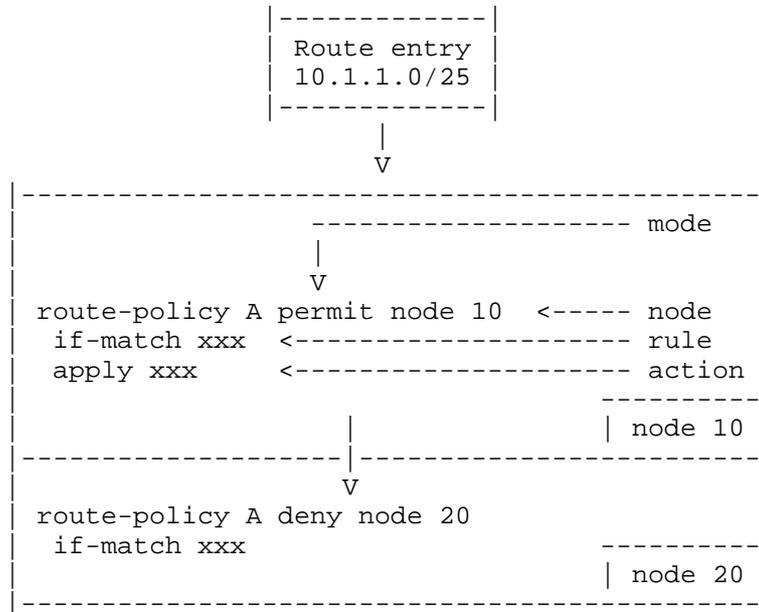
4.8. IPv6 Prefix Filters Configuration

An IPv6 prefix list contains a group of route filtering rules. Users can specify the prefix and mask length range to match the destination network segment address or the next hop address of a route. An IPv6 prefix list is used to filter routes that are advertised and received by various dynamic routing protocols..

```
+--rw ipv6PrefixFilters
|  +--rw ipv6PrefixFilter* [name]
|  |   +--rw name                string
|  |   +--ro permitCnt?          uint32
|  |   +--ro denyCnt?            uint32
|  |   +--rw ipv6PrefixFilterNodes
|  |   |   +--rw ipv6PrefixFilterNode* [nodeSequence]
|  |   |   |   +--rw nodeSequence    uint32
|  |   |   |   +--rw matchMode?      enumeration
|  |   |   |   +--rw address          inet:ipv6-address
|  |   |   |   +--rw maskLength      uint8
|  |   |   |   +--rw matchNetwork?   boolean
|  |   |   |   +--rw greaterEqual?   uint8
|  |   |   |   +--rw lessEqual?      uint8
```

4.9. Route-Policy Configuration

As shown in following figure, a Route-Policy consists of node IDs, matching mode, if-match clauses, and apply clauses.



Composition of a Route-Policy

o Node ID

A Route-Policy consists of one or more nodes. Node IDs are specified as indexes in the IP prefix list. In a Route-Policy, routes are filtered based on the following rules:

- 1) Sequential matching: The system checks entries based on node IDs in ascending order. Therefore, specifying the node IDs in the required sequence is recommended.
- 2) One-time matching: The relationship between the nodes of a Route-Policy is "OR". If a route matches one node, the route matches the Route-Policy and will not be matched against the next node.

o Matching mode

Either of the following matching modes can be used:

- 1) permit: specifies the permit mode of a node. If a route matches the if-match clauses of a node, all the actions defined by apply

clauses are performed, and the matching is complete. If a route does not match the if-match clauses of the node, the route continues to match the next node.

2) deny: specifies the deny mode of a node. In the deny mode, the apply clauses are not used. If a route matches all the if-match clauses of the node, the route is denied by the node and the next node is not matched. If the entry does not match all the if-match clauses, the next node is matched.

o if-match clause

The if-match clause defines the matching rules. Each node of a Route-Policy can comprise multiple if-match clauses or no if-match clause at all. If no if-match clause is configured for a node in the permit mode, all routes match the node.

o apply clause

The apply clauses specify actions. When a route matches a Route-Policy, the system sets some attributes for the route based on the apply clause. Each node of a Route-Policy can comprise multiple apply clauses or no apply clause at all. The apply clause is not used when routes need to be filtered but attributes of the routes do not need to be set.

Matching results of a Route-Policy

The matching results of a Route-Policy are obtained based on the following aspects:

- 1) Matching mode of the node, either permit or deny
- 2) Matching rules (either permit or deny) contained in the if-match clause (such as ACLs or IP prefix lists)

A Route-Policy is a complex filter. It is used to match attributes of specified routes and change route attributes when specific conditions are met. A Route-Policy can use the preceding filters to define its matching rules.

```

+--rw routePolicys
  +--rw routePolicy* [name]
    +--rw name string
    +--rw routePolicyNodes
      +--rw routePolicyNode* [nodeSequence]
        +--rw nodeSequence uint32
        +--rw matchMode? enumeration
        +--rw description? string
        +--ro matchCount? uint32
        +--rw matchCondition
          | +--rw matchCostValue uint32
          | ...
          | +--rw matchExtCmntySooList? string
        +--rw applyAction
          +--rw applyAsPaths
          ...
          +--rw applyPriorityValue uint16

```

5. Route Policy Yang Module

ROUTING POLICY YANG MODEL

<CODE BEGINS> file "routing-policy@2014-12-12.yang"

```

module routing-policy {
  namespace "urn:huawei:params:xml:ns:yang:routing-policy";
  // replace with IANA namespace when assigned
  prefix "routing-policy";

  import ietf-interfaces {
    prefix if;
    //rfc7223-YANG Interface Management
  }

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

  description
    "This YANG module defines the RTP configuration data for
    routing policy service.

    Terms and Acronyms

    RTP: Routing-Policy
    Routing policies are used to filter routes and set attributes
    for routes. Changing route attributes (including reachability)
    changes the path that network traffic passes through."

```



```
    }//End of grouping basicCommunityNodes

    grouping advancedCommunityNodes {
        container advancedCommunityNodes {

            list advancedCommunityNode {
                key "nodeSequence matchMode";
                min-elements "0";
                max-elements "unbounded";

                leaf nodeSequence {
                    config "true";
                    type uint32 {
                        range "1..4294967295";
                    }
                }
                uses matchMode;

                leaf regular {
                    description
                    "The ip community-filter advanced comm-filter-name command
                    or the ip community-filter adv-comm-filter-num command can
                    be used to configure an advanced community filter. advanced
                    comm-filter-name specifies the name of an advanced
                    community filter, and the name cannot be all digits.
                    adv-comm-filter-num specifies only the advanced community
                    filter with the number ranging from 100 to 199.";

                    config "true";
                    mandatory "true";
                    type "string";
                }
            }
        }
    }//End of grouping advancedCommunityNodes

    grouping basicExtendCommunityFilterNodes {
        container extendCommunityFilterNodes {

            list extendCommunityFilterNode {
                key "nodeSequence";
                min-elements "0";
                max-elements "unbounded";

                leaf nodeSequence {
                    config "true";
                    type uint32 {
```



```
/*
  IP Routing Policy Configuration Commands.

  Routing policies are implemented using the following procedures:
  1)Define rules:
  Define features of routes to which routing policies are applied.
  Users define a set of matching rules based on different
  attributes of routes, such as the destination address and the
  address of the router that advertises the routes.
  2)Implement the rules:
  Apply the matching rules to routing policies for advertising,
  receiving, and importing routes.
  3)Filter
  A filter is the core of a routing policy and is defined using a
  set of matching rules.
*/
container asPathFilters {
  description
    "An AS_Path filter uses the regular expression to define
    matching rules.";

  list asPathFilterKeyedByIndex {
    key "index";
    min-elements "0";
    max-elements "unbounded";

    leaf index {
      config "true";
      type uint32 {
        range "1..256";
      }
    }
  }
  uses asPathFilterNodes;
}

list asPathFilterKeyedByName {
  key "name";
  min-elements "0";
  max-elements "unbounded";

  leaf name {
    config "true";
    type "string";
  }
  uses asPathFilterNodes;
}
} //End of container asPathFilters
```

```
container communityFilters {
  description
    "The ip community-filter command configures a community filter
    or one entry in the community filter.

    The community attribute is a private attribute of BGP, and can
    be used only to filter BGP routes. The community attribute can
    be used together with the if-match community-filter command as
    a matching condition of a route-policy.";

  list basicCommunityFilterKeyedByIndex {
    key "index";
    min-elements "0";
    max-elements "unbounded";

    leaf index {
      config "true";
      type uint32 {
        range "1..99";
      }
    }
    uses basicCommunityNodes;
  }

  list basicCommunityFilterKeyedByName {
    key "name";
    min-elements "0";
    max-elements "unbounded";

    leaf name {
      config "true";
      type "string";
    }
    uses basicCommunityNodes;
  }

  list advancedCommunityFilterKeyedByIndex {
    key "index";
    min-elements "0";
    max-elements "unbounded";

    leaf index {
      config "true";
      type uint32 {
        range "100..199";
      }
    }
    uses advancedCommunityNodes;
  }
}
```

```
    }

    list advancedCommunityFilterKeyedByName {
      key "name";
      min-elements "0";
      max-elements "unbounded";

      leaf name {
        config "true";
        type "string";
      }
      uses advancedCommunityNodes;
    }
  } //End of container communityFilters

  container extendCommunityFilters {
    description
      "The ip extcommunity-filter command adds an extended community
      filter.

      An extended community filter can be used as a matching
      condition of a route-policy by using a command such as
      if-match extcommunity-filter zz.";

    list basicExtendCommunityFilterKeyedByIndex {
      key "index";
      min-elements "0";
      max-elements "unbounded";

      leaf index {
        config "true";
        type uint32 {
          range "1..199";
        }
      }
      uses basicExtendCommunityFilterNodes;
    }
    list basicExtendCommunityFilterKeyedByName {
      key "name";
      min-elements "0";
      max-elements "unbounded";

      leaf name {
        config "true";
        type "string";
      }
      uses basicExtendCommunityFilterNodes;
    }
  }
}
```

```
    }
    list advancedExtendCommunityFilterKeyedByIndex {
      key "index";
      min-elements "0";
      max-elements "unbounded";

      leaf index {
        config "true";
        type uint32 {
          range "200..399";
        }
      }
      uses advancedExtendCommunityNodes;
    }
    list advancedExtendCommunityFilterKeyedByName {
      key "name";
      min-elements "0";
      max-elements "unbounded";

      leaf name {
        config "true";
        type "string";
      }
      uses advancedExtendCommunityNodes;
    }
  } //End of container extendCommunityFilters

  container extendCommunitySooLists {
    description
      "The ip extcommunity-list soo command configures a Source of
      Origin (SoO) extended community filter.

      SoO records the BGP route originator. To configure an SoO
      extended community filter so that BGP routes carrying SoO
      can be filtered, run the ip extcommunity-list soo command.";

    list extendCommunitySooList {
      key "name";
      min-elements "0";
      max-elements "unbounded";

      leaf filterType {
        config "true";
        mandatory "true";
        type enumeration {
```

```
        enum "basic" {
            value "0";
            description "basic:";
        }
        enum "advanced" {
            value "1";
            description "advanced:";
        }
    }
}
leaf name {
    config "true";
    type "string";
}

container basicExtendCommunitySooListNodes {
    list basicExtendCommunitySooListNode {
        key "nodeSequence";
        min-elements "0";
        max-elements "unbounded";

        leaf nodeSequence {
            config "true";
            type uint32 {
                range "1..4294967295";
            }
        }
    }
    uses matchMode;

    container extendEomunityNumbers {
        list extendCommunityNumber {
            key "extendCommunityNumberValue";
            min-elements "1";
            max-elements "16";

            leaf extendCommunityNumberValue {
                config "true";
                type string;
            }
        }
    }
}

container advanceExtendEomunitySooListNodes {
    list advanceExtendEomunitySooListNode {
        key "nodeSequence matchMode";
        min-elements "0";
    }
}
```

```
        max-elements "unbounded";

        leaf nodeSequence {
            config "true";
            type uint32 {
                range "1..4294967295";
            }
        }

        uses matchMode;

        leaf regular {
            config "true";
            mandatory "true";
            type "string";
        }
    }
}

} // End of container extendCommunitySooLists

container rdFilters {
    description
        "The ip rd-filter command configures a route distinguisher (RD)
        filter.

        The RD attribute is carried in VPN routes. RDs are used to
        distinguish address spaces with the same IPv4 address prefix.
        An RD filter is used to filter VPN routes. The VPN target
        attribute of the VPN route controls the route exchange between
        VPN instances. The RD filter can filter a VPN route or multiple
        VPN routes from VPN instances.";

    list rdFilter {
        key "index";
        min-elements "0";
        max-elements "unbounded";

        leaf index {
            config "true";
            type uint32 {
                range "1..199";
            }
        }
    }
}
```

```
    container rdFilterNodes {
      list rdFilterNode {
        key "nodeSequence";
        min-elements "0";
        max-elements "unbounded";

        leaf nodeSequence {
          config "true";
          type uint32 {
            range "1..4294967295";
          }
        }
      }

      uses matchMode;

      container rdStrings {
        list rdString {
          key "rdStringValue";
          min-elements "0";
          max-elements "unbounded";

          leaf rdStringValue {
            config "true";
            type "string";
          }
        }
      }
    }
  }
} //End of container rdFilters
```

```
container prefixFilters {
  description
    "The ip ip-prefix command configures a new IP prefix list or
    one entry in an existing IP prefix list.

    An IP prefix list can be used as a filter or as matching
    conditions of a routing policy when it is used together with
    the if-match command.";

  list prefixFilter {
    key "name";
    min-elements "0";
    max-elements "unbounded";
```

```
leaf name {
  config "true";
  type "string";
}

leaf permitCnt {
  config "false";
  default "0";
  type uint32 {
    range "0..65535";
  }
}

leaf denyCnt {
  config "false";
  default "0";
  type uint32 {
    range "0..65535";
  }
}

container prefixFilterNodes {
  list prefixFilterNode {
    key "nodeSequence";
    min-elements "0";
    max-elements "unbounded";

    leaf nodeSequence {
      config "true";
      type uint32 {
        range "1..4294967295";
      }
    }
  }

  uses matchMode;

  leaf address {
    config "true";
    mandatory "true";
    type inet:ipv4-address;
  }

  leaf maskLength {
    config "true";
    mandatory "true";
    type uint8 {
      range "0..32";
    }
  }

  leaf matchNetwork {
    config "true";
  }
}
```



```
leaf denyCnt {
  config "false";
  default "0";
  type uint32 {
    range "0..65535";
  }
}
container ipv6PrefixFilterNodes {
  list ipv6PrefixFilterNode {
    key "nodeSequence";
    min-elements "0";
    max-elements "unbounded";

    leaf nodeSequence {
      config "true";
      type uint32 {
        range "1..4294967295";
      }
    }
  }

  uses matchMode;

  leaf address {
    config "true";
    mandatory "true";
    type inet:ipv6-address;
  }
  leaf maskLength {
    config "true";
    mandatory "true";
    type uint8 {
      range "0..128";
    }
  }
  leaf matchNetwork {
    config "true";
    default "false";
    type "boolean";
  }
  leaf greaterEqual {
    config "true";
    type uint8 {
      range "0..128";
    }
  }
  leaf lessEqual {
    config "true";
    type uint8 {
```

```
        range "0..128";
      }
    }
  }
}

}
} //End of container ipv6PrefixFilters

/*
Routing policies are used to filter routes and set attributes for
routes. Changing route attributes (including reachability) changes
the path that network traffic passes through.
*/
container routePolicys {
  list routePolicy {
    key "name";
    min-elements "0";
    max-elements "unbounded";

    leaf name {
      config "true";
      type "string";
    }
  }

  /*
A route-policy is used to filter routes and set route attributes
for the routes that match the route-policy. A route-policy
consists of multiple nodes. One node can be configured with
multiple if-match and apply clauses.
*/
  container routePolicyNodes {
    list routePolicyNode {
      key "nodeSequence";
      min-elements "0";
      max-elements "unbounded";

      leaf nodeSequence {
        config "true";
        type uint32 {
          range "0..65535";
        }
      }
    }

    uses matchMode;

    leaf description {
```

```
description
  "The description command configures the description of
  a route-policy.";
config "true";
mandatory "false";
type "string";
}

leaf matchCount {
  config "false";
  mandatory "false";
  type "uint32";
}

/*if-match clauses*/
container matchCondition {
  description
    "Define a set of matching rules and setting rules.
    The policy is applied to the routing information to
    meet the requirements of the matching rules.";

  leaf matchCostValue {
    description
      "The if-match cost command sets a matching rule that
      is based on the route cost.

      You can use the if-match cost command to configure
      a node to filter routes based on the route costs.
      After such a matching rule is configured, you can
      apply the apply clauses to change the attributes of
      the routes that match the matching rule.";

    config "true";
    mandatory "true";
    type uint32 {
      range "0..4294967295";
    }
  } //End of leaf matchCostValue

  container matchInterfaces {
    description
      "The if-match interface command sets a matching rule
      that is based on the outbound interface.

      The if-match interface command is used to filter
      routes based on the outbound interfaces.
      A maximum of 16 outbound interfaces can be
      configured in this command. ";
```

```
list matchInterface {
  key "ifName";
  min-elements "0";
  max-elements "unbounded";

  leaf ifName {
    config "true";
    type leafref {
      path "/if:interfaces/if:interface/if:name";
    }
  }
}
} //End of container matchInterfaces

container matchRouteTypes {
  description
    "The if-match route-type command sets a filtering
    rule that is based on the route type.

    To filter OSPF or IS-IS routes based on the route
    type, run the if-match route-type command.";

  list matchRouteType {
    key "routeType";
    min-elements "0";
    max-elements "unbounded";

    leaf routeType {
      config "true";
      type enumeration {
        enum "external1" {
          value "0";
          description "external1:";
        }
        enum "external2" {
          value "1";
          description "external2:";
        }
        enum "internal" {
          value "2";
          description "internal:";
        }
        enum "isisLevel1" {
          value "3";
          description "isisLevel1:";
        }
        enum "isisLevel2" {
          value "4";
        }
      }
    }
  }
}
```



```
        "The if-match acl command sets a matching rule
        that is based on the Access Control List (ACL).";

        config "true";
        mandatory "true";
        type "string";
    }

    leaf matchDestPrefixFilter {
        description
            "The if-match ip-prefix command sets a matching
            rule that is based on the IP prefix list.";

        config "true";
        mandatory "true";
        type "string";
    }

    leaf matchDestPrefix6Filter {
        description
            "Specify the ipv6-prefix name to be matched";

        config "true";
        mandatory "true";
        type "string";
    }

    leaf matchDestAcl6 {
        description
            "Specify the ipv6 acl name or num.";

        config "true";
        mandatory "true";
        type "string";
    }

    container matchIPv4NextHop {
        description
            ".";

        choice matchIPv4NextHopMode {
            case matchNextHopPrefixFilter {
                leaf prefixName {
                    config "true";
                    mandatory "true";
                    type "string";
                }
            }
        }
    }
}
```

```
    case matchNextHopAcl {
      leaf aclNameOrNum {
        config "true";
        mandatory "true";
        type "string";
      }
    }
  }
}

container matchIPv6NextHop {
  choice matchIPv6NextHopMode {
    case matchNextHopPrefix6Filter {
      leaf ipv6PrefixName {
        config "true";
        mandatory "true";
        type "string";
      }
    }
    case matchNextHopAcl6 {
      leaf aclNameOrNum {
        config "true";
        mandatory "true";
        type "string";
      }
    }
  }
}

container matchIPv4RouteSource {
  choice matchIPv4RouteSourceMode {
    case matchRtSrcPrefixFilter {
      leaf prefixName {
        config "true";
        mandatory "true";
        type "string";
      }
    }
    case matchRouteSourceAcl {
      leaf aclNameOrNum {
        config "true";
        mandatory "true";
        type "string";
      }
    }
  }
}
```

```
container matchIPv6RouteSource {
  choice matchIPv6RouteSourceMode {
    case matchRtSrcPrefix6Filter {
      leaf ipv6PrefixName {
        config "true";
        mandatory "true";
        type "string";
      }
    }
    case matchRtSrcAcl6 {
      leaf aclNameOrNum {
        config "true";
        mandatory "true";
        type "string";
      }
    }
  }
}

container matchCommunityFilters {
  description
    "The if-match community-filter command sets a
    matching rule that is based on the community
    filter.";

  list matchCommunityFilter {

    key "cmntyNameOrNum";
    min-elements "0";
    max-elements "unbounded";

    leaf wholeMatch {
      config "true";
      default "false";
      type "boolean";
    }
    leaf cmntyNameOrNum {
      config "true";
      type "string";
    }
  }
}

container matchExtcommunityFilters {
  description
    "The if-match extcommunity-filter command sets a
    matching rule that is based on the extended
    community filter.";
```

```
list matchExtcommunityFilterIndex {
  key "extCmntyIndex";
  min-elements "0";
  max-elements "unbounded";

  leaf extCmntyIndex {
    config "true";
    type uint32 {
      range "1..399";
    }
  }
}
list matchExtcommunityFilterName {

  key "extCmntyName";
  min-elements "0";
  max-elements "unbounded";

  leaf extCmntyName {
    config "true";
    type "string";
  }
}
}

leaf matchRdFilter {
  description
    "The if-match rd-filter command sets a matching rule
    that is based on the Route Distinguisher (RD)
    filter.";

  config "true";
  mandatory "true";
  type uint32 {
    range "1..199";
  }
}

container matchAsPathFilters {
  description
    "The if-match as-path-filter command sets a matching
    rule that is based on the AS_Path filter.";

  list matchAsPathFilterIndex {
    key "asPathIndex";
    min-elements "0";
    max-elements "unbounded";
```

```
    leaf asPathIndex {
      config "true";
      type uint32 {
        range "1..256";
      }
    }
  }
}
list matchAsPathFilterName {
  key "asPathFilterName";
  min-elements "0";
  max-elements "unbounded";

  leaf asPathFilterName {
    config "true";
    type "string";
  }
}
}
```

```
leaf matchOriginAsValidateResult {
  description
    "The if-match rpki origin-as-validation command
    configures a filtering rule based on the BGP
    origin AS validation result.
```

Attackers can steal user data by advertising routes that are more specific than those advertised by carriers. RPKI can address this issue by validating the origin ASs of BGP routes and apply the BGP origin AS validation result to route selection. The validation result can be Valid, Not Found, or Invalid.

To configure a filtering rule based on the BGP origin AS validation result, you can run the if-match rpki origin-as-validation command. After the filtering rule is configured, attribute of the routes that match the filtering rule can be modified based on the apply clause."

```
config "true";
mandatory "true";
type enumeration {
  enum "valid" {
    value "0";
    description "valid:";
  }
  enum "invalid" {
```

```
        value "1";
        description "invalid:";
    }
    enum "notFound" {
        value "2";
        description "notFound:";
    }
}
} //End of leaf matchOriginAsValidateResult

leaf matchExtCmntySooList {
    description
        "The if-match extcommunity-list soo command sets a
        filtering rule that is based on the Source of
        Origin (SoO) extended community filter.

        The extended community attributes help flexibly
        control the route-policy. You can use the if-match
        extcommunity-list soo command to configure a node
        to filter routes based on the SoO extended
        community filter.";

    config "true";
    type "string";
}

} //End of container matchCondition

/*apply clauses*/
container applyAction {
    description
        "Apply the matching rules to the routing policies for
        route advertisement, reception, and import.";

    container applyAsPaths {
        description
            "The apply as-path command replaces the original
            AS_Path list or add the specified AS number to
            the original AS_Path list when BGP route selection
            is adjusted.";
        container asPathStrings {
            list asPathString {
                key "sequenceNumber";
                min-elements "0";
                max-elements "unbounded";

                leaf sequenceNumber {
```

```
        config "true";
        type uint32 {
            range "1..10";
        }
    }
    leaf stringValue {
        config "true";
        mandatory "true";
        type "string";
    }
}
}
leaf operationType {
    config "true";
    type enumeration {
        enum "delete" {
            value "0";
            description "delete:";
        }
        enum "replace" {
            value "1";
            description "replace:";
        }
        enum "additive" {
            value "2";
            description "additive:";
        }
        enum "delSpecial" {
            value "3";
            description "delSpecial:";
        }
    }
    default "replace";
}
} //End of container applyAsPaths

container applyNextHops {
    description
    "The apply ip-address next-hop command sets the
    next hop address of a route.
    In BGP, the next hop address of a route can be
    set through the route-policy in the following
    situations:
    1) IBGP: For an IBGP peer, the configured import
    and export policies can take effect. If the
    next hop address configured in the policy is
    unreachable, the IBGP peer still adds the route
    to the BGP routing table, but the route is not
```

valid.

- 2) EBGP: For an EBGP peer, when the policy is used to modify the next hop address of a route, the import policy is configured. If the export policy is configured, the route is discarded because its next hop is unreachable. When the EBGP peer relationship is established through a physical connection, the policy cannot take effect. That is, the next hop address of the route cannot be modified.";

```
leaf nextHop {
  config "true";
  mandatory "true";
  type inet:ipv4-address;
}
leaf isPeerAddress {
  config "true";
  type "boolean";
  default "false";
}
} //End of container applyNextHops

leaf applyLocalPreference {
  description
    "The apply local-preference command sets the local
    preference (Local-Pref) for BGP routes.

    The Local-Pref attribute is the proprietary
    attribute of BGP. Therefore, the apply
    local-preference command sets only the Local-Pref
    for BGP routes. The Local_Pref attribute is used
    to determine the optimal route when traffic leaves
    an AS. When a BGP router obtains multiple routes to
    the same destination address but with different
    next hops through IBGP peers, the route with the
    largest Local_Pref value is selected.";

  config "true";
  mandatory "true";
  type uint32 {
    range "0..4294967295";
  }
}

container applyCosts {
  description
    "The apply cost command sets the MED value for BGP
```

routes or cost value for routes of other protocols. When the filtering conditions specified by if-match clauses are met, you can run the apply cost command to change the route MED or cost to control route selection. After setting the MED or cost, the MED or cost of the routes that are imported using the route-policy is changed accordingly.";

```
leaf applyChoice {
  config "true";
  mandatory "true";
  type enumeration {
    enum "Add" {
      value "0";
      description "Add:";
    }
    enum "Sub" {
      value "1";
      description "Sub:";
    }
    enum "Replace" {
      value "2";
      description "Replace:";
    }
    enum "Inherit" {
      value "3";
      description "Inherit:";
    }
  }
}
leaf costValue {
  config "true";
  mandatory "true";
  type uint32 {
    range "0..4294967295";
  }
}
} //End of container applyCosts

container applyIpv6NextHops {
  description
    "The apply ipv6 next-hop command configures an IPv6
    next hop address for a BGP route through a
    route-policy.
    In BGP, the next hop address of a route can be set
    through the route-policy in the following
    situations:
    1) IBGP: For an IBGP peer, the configured inbound
```

and outbound policies can take effect. If the next hop address configured in the policy is unreachable, the IBGP peer still adds the route to the BGP routing table, but the route is not valid.

- 2) EBGP: For an EBGP peer, when the policy is used to modify the next hop address of a route, the inbound policy is configured. If the outbound policy is configured, the route is discarded because its next hop is unreachable. When the EBGP peer relationship is established through a physical connection, the policy cannot take effect. That is, the next hop address of the route cannot be modified.";

```
leaf ipv6NextHop {
  config "true";
  mandatory "true";
  type inet:ipv6-address;
}
leaf isPeerAddress {
  config "true";
  type "boolean";
  default "false";
}
} //End of container applyIpv6NextHops

leaf applyCostType {
  description
    "The apply cost-type command sets the cost type of
    the route.
    1) The apply cost-type { external | internal }
    command sets the cost type of IS-IS routes. The
    cost of an internal route imported to IS-IS remains
    unchanged and the cost of an external route imported
    to IS-IS is increased by 64.
    2) The apply cost-type { type-1 | type-2 }
    command modifies the type of OSPF routes. During
    route import, OSPF modifies the type but not the
    cost value of the original route. When OSPF
    advertises the imported route with the cost and
    type information to a peer, the peer device will
    recalculate the cost value of the imported
    route based on the received information.";

  config "true";
  mandatory "true";
  type enumeration {
```

```
enum "external" {
  value "0";
  description "external:";
}
enum "internal" {
  value "1";
  description "internal:";
}
enum "type_1" {
  value "2";
  description "type_1:";
}
enum "type_2" {
  value "3";
  description "type_2:";
}
}
} //End of leaf applyCostType

container applyOrigin {
  description
    "The apply origin command sets the Origin attribute
    of BGP routes. The Origin attribute, as a
    proprietary attribute of BGP, defines the origin of
    a route. It identifies how a BGP route is generated.
    You can use the apply origin command to set the
    Origin attribute of BGP routes.";

  leaf originType {
    config "true";
    mandatory "true";
    type enumeration {
      enum "egp" {
        value "0";
        description "egp:";
      }
      enum "igp" {
        value "1";
        description "igp:";
      }
      enum "incomplete" {
        value "2";
        description "incomplete:";
      }
    }
  }
  leaf asStrOrNum {
    config "true";
  }
}
```

```
        mandatory "true";
        type "string";
    }
} //End of container applyOrigin

container applyCommunitys {
    description
        "The apply community command configures BGP
        community attributes. The community attribute,
        which is the private attribute of BGP, simplifies
        the application of routing policies and facilitates
        route maintenance and management. A community is a
        set of destination addresses with the same
        characteristics. These addresses have no physical
        boundary and are independent of their ASs. They
        share one or multiple community attributes,
        which can be changed or set through the apply
        community command.";

    leaf operationType {
        config "true";
        default "replace";
        type enumeration {
            enum "delete" {
                value "0";
                description "delete:";
            }
            enum "replace" {
                value "1";
                description "replace:";
            }
            enum "additive" {
                value "2";
                description "additive:";
            }
            enum "delSpecial" {
                value "3";
                description "delSpecial:";
            }
        }
    }
}

container applyCmntyStrings {
    list applyCmntyString {
        key "stringValue";
        min-elements "0";
        max-elements "unbounded";
    }
}
```

```
        leaf stringValue {
            config "true";
            type "string";
        }
    }
}
} //End of container applyCommunitys

leaf applyTagValue {
    description
        "The apply tag command sets the tag for routes.
        If routes satisfy the filter condition specified
        by the if-match clause, you can run the apply tag
        command to set the same tag for the routes that
        satisfy the same filter condition to classify the
        routes.";

    config "true";
    mandatory "true";
    type uint32 {
        range "0..4294967295";
    }
}

leaf applyMplsLabel {
    description
        "The apply mpls-label command assigns MPLS labels to
        routes. In the scenario where inter-AS VPN Option C
        or carrier's carrier (CSC) is deployed, you can use
        the apply mpls-label command to allocate labels to
        public routes.";

    config "true";
    mandatory "true";
    type "boolean";
}

leaf applyPreference {
    description
        "The apply preference command sets the preference
        for routes. If a route satisfies the filter
        condition specified by the if-match clause, you
        can run the apply preference command to change
        the preference of the route to participate in
        route selection. Then, when different protocols
        discover multiple routes to the same destination,
        the route discovered by the protocol with a higher
        preference is selected as a valid route to forward
```

```
        packets.";

        config "true";
        mandatory "true";
        type uint32 {
            range "1..255";
        }
    }

    container applyExtendCommunitys {
        description
            "The apply extcommunity command configures extended
            community attributes for BGP routes.
            The apply extcommunity command is applicable to
            BGP/MPLS IP VPNs. At present, there are two types
            of BGP extended community attributes.
            1) VPN route-target (RT) extended community
            2) Source of Origin (SoO) extended community
            At present, RT extended community attributes can be
            set only through the route-policy. This command
            cannot specify an extended community attribute for
            public routes.";

        container applyExtCmntyStrings {
            list applyExtCmntyString {
                key "stringValue";
                min-elements "0";
                max-elements "unbounded";

                leaf stringValue {
                    config "true";
                    type "string";
                }
            }
        }

        leaf additiveFlag {
            config "true";
            default "false";
            type "boolean";
        }
    } //End of container applyExtendCommunitys

    container applyCommunityFilterDelete {
        description
            "The apply comm-filter delete command deletes a BGP
            route community according to the specified value in
            the community filter. The community filter can be
```

either a basic or advanced community filter.

When the `apply comm-filter delete` command is run in the Route-Policy view to delete the values in the community filter, only one community attribute can be specified in an `ip community-filter` command. If multiple community attributes are configured in the same community filter, running the `apply comm-filter delete` command cannot delete these community attributes. To delete the community attributes, you need to run the `ip community-filter` command several times to configure community attributes one by one, and then run the `apply comm-filter delete` command to delete these community attributes.";

```
leaf communityNum {
  config "true";
  type "string";
}
leaf communityName {
  config "true";
  type "string";
}
} //End of container applyCommunityFilterDelete

container applyDampening {
  description
    "The apply dampening command sets dampening
    parameters for EBGp routes. The apply dampening
    command, which is mostly used in BGP, is used
    to prevent frequent route dampening from
    affecting routers on the network.
    You can configure different route dampening
    parameters for different nodes in the same
    route-policy. When route flapping occurs, BGP
    can use different route dampening parameters
    to suppress the routes that match the
    route-policy.";

  leaf halfLifeValue {
    config "true";
    mandatory "true";
    type uint32 {
      range "1..45";
    }
  }
  leaf reuseValue {
    config "true";
  }
}
```

```
        mandatory "true";
        type uint32 {
            range "1..20000";
        }
    }
    leaf suppressValue {
        config "true";
        mandatory "true";
        type uint32 {
            range "1..20000";
        }
    }
    leaf ceilingValue {
        config "true";
        mandatory "true";
        type uint32 {
            range "1001..20000";
        }
    }
} //End of container applyDampening

leaf applyRouteType {
    description
        "
        1) After a route matches a route-policy, you can
            set the level of the route imported to IS-IS.
        2) Routes matching the if-match clauses defined
            in the routing policy will be imported into
            the OSPF area specified in the command.";

    config "true";
    mandatory "true";
    type enumeration {
        enum "OspfStubArea" {
            value "0";
            description "OspfStubArea:";
        }
        enum "OspfBackbone" {
            value "1";
            description "OspfBackbone:";
        }
        enum "IsisLevel1" {
            value "2";
            description "IsisLevel1:";
        }
        enum "IsisLevel2" {
            value "3";
            description "IsisLevel2:";
        }
    }
}
```

```
    }
    enum "IsisLevel12" {
        value "4";
        description "IsisLevel12:";
    }
}
} //End of leaf applyRouteType

leaf applyPreferredValue {
    description
        "The apply preferred-value command sets the preferred
        value for BGP routes.
        The preferred value is a proprietary attribute of
        BGP.
        You can use the apply preferred-value command to set
        the preferred value for a BGP route in the import
        policy.";

    config "true";
    mandatory "true";
    type uint32 {
        range "0..65535";
    }
} //End of leaf applyPreferredValue

container applyQosParas {
    leaf QosType {
        config "true";
        mandatory "true";
        type enumeration {
            enum "QosID" {
                value "0";
                description "qosLocalID:";
            }
            enum "Behavior" {
                value "1";
                description "behavior:";
            }
            enum "IpPrecedence" {
                value "2";
                description "ipPrecedence:";
            }
        }
    }
}

leaf qosLocalID {
    description
        "The apply qos-local-id command sets the QoS local
        ID."
```

The QoS local ID is a local identifier of QoS. In actual applications, you can set the QoS local ID in the route-policy, and add the command that matches the QoS local ID in the QoS policy. The QoS local ID set in the route-policy is delivered to the FIB table. During packet forwarding, the system obtains the QoS local ID from the FIB table and applies the related QoS policy according to the QoS local ID."

```
config "true";
mandatory "true";
type uint32 {
  range "1..4095";
}
}

leaf ipPrecedence {
  description
    "The apply ip precedence command sets the QoS
    parameter ip-precedence for routes.

    The apply ip precedence command is generally
    applicable to QPPB, in which the IP precedence
    is a QoS parameter that can be set. After
    receiving routes, a BGP route receiver matches
    the attributes of the BGP routes based on the
    import route-policy, sets the IP precedence,
    delivers the BGP routes together with the
    associated QoS parameters, and applies QoS
    traffic policies to the classified data. In
    this case, the BGP route receiver can apply
    QoS policies to the data sent to the destination
    network segment based on the IP precedence. This
    applies QoS policies in BGP.";

  config "true";
  mandatory "true";
  type uint32 {
    range "0..7";
  }
}

leaf behaviorName {
  description
    "The apply behavior command configures a QoS
    traffic behavior for routes.";
```

```
        config "true";
        mandatory "true";
        type "string";
    }
} //End of container applyQosParas

leaf applyTrafficIndex {
    description
        "The apply traffic-index command sets the BGP
        traffic index. BGP accounting uses different
        BGP traffic indexes in BGP community
        attributes to identify routes and charge the
        traffic accordingly.";

    config "true";
    mandatory "true";
    type uint32 {
        range "1..64";
    }
} //End of leaf applyTrafficIndex

container applyExtCmntySoos {
    description
        "The apply extcommunity soo command configures Source
        of Origin (SoO) extended community attributes for
        BGP routes.";

    leaf operationType {
        config "true";
        type enumeration {
            enum "delete" {
                value "0";
                description "delete:";
            }
            enum "replace" {
                value "1";
                description "replace:";
            }
            enum "additive" {
                value "2";
                description "additive:";
            }
            enum "delSpecial" {
                value "3";
                description "delSpecial:";
            }
        }
        default "replace";
    }
}
```

```
    }
    container applyExtCmntySooStrings {
      list applyExtCmntySooString {
        key "stringValue";
        min-elements "0";
        max-elements "unbounded";

        leaf stringValue {
          config "true";
          type "string";
        }
      }
    }
  } //End of container applyExtCmntySoos

  leaf applyPriorityValue {
    description
      "The apply preference command sets the preference
      for routes.

      If a route satisfies the filter condition specified
      by the if-match clause, you can run the apply
      preference command to change the preference of the
      route to participate in route selection. Then, when
      different protocols discover multiple routes to the
      same destination, the route discovered by the
      protocol with a higher preference is selected as a
      valid route to forward packets.";

    config "true";
    mandatory "true";
    type uint16 {
      range "1..255";
    }
  } //End of leaf applyPriorityValue
}
} //End of container routePolicyNodes
}
} //End of container routePolicys
} //End of container routing-policy
}
<CODE ENDS>
```

6. IANA Considerations

This document makes no request of IANA.

7. Security Considerations

This document does not introduce any new security risk.

8. Acknowledgements

The authors would like to thank Xianping Zhang, Nan Meng, Linghai Kong for their contributions to this work.

9. 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.
- [RFC6241] Enns, R., Bjorklund, M., Schoenwaelder, J., and A. Bierman, "Network Configuration Protocol (NETCONF)", RFC 6241, June 2011.

Authors' Addresses

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