Abstract

SR P2MP policies are set of policies that enable architecture for P2MP service delivery.

A P2MP policy consists of candidate paths that connects the Root of the Tree to a set of Leaves. The P2MP policy is composed of replication segments. A replication segment is a forwarding instruction for a candidate path which is downloaded to the Root, transit nodes and the leaves.

This document defines a YANG data model for SR P2MP Policy Configuration and operation.

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.
1. Introduction

This document defines a YANG data model for P2MP SR Policy configuration and operation.
The draft [draft-voyer-spring-sr-p2mp-policy] defines a variant of the SR Policy [I-D. ietf-spring-segment-routing-policy] for constructing a P2MP segment to support multicast service delivery.

A Point-to-Multipoint (P2MP) Policy contains a set of candidate paths and identifies a Root node and a set of Leaf nodes in a Segment Routing Domain. The draft also defines a Replication segment, which corresponds to the state of a P2MP segment on a particular node. The Replication segment is the forwarding instruction for a P2MP LSP at the Root, Transit and Leaf nodes.

For a P2MP segment, a controller may be used to compute a tree from a Root node to a set of Leaf nodes, optionally via a set of replication nodes. A packet is replicated at the root node and optionally on Replication nodes towards each Leaf node.

We define two types of a P2MP segment: Spray and Replication.

A Point-to-Multipoint service delivery could be via Ingress Replication (aka Spray in some SR context), i.e., the root unicasts individual copies of traffic to each leaf. The corresponding P2MP segment consists of replication segments only for the root and the leaves.

A Point-to-Multipoint service delivery could also be via Downstream Replication (aka TreeSID in some SR context), i.e., the root and some downstream replication nodes replicate the traffic along the way as it traverses closer to the leaves.

It should be noted that two replication nodes can be connected directly, or they can be connected via unicast SR segment or a segment list.

The leaves and the root of a p2mp policy can be discovered via the NG-MVPN procedures [RFC 6513 and RFC 6514] or manually configured.

Base on the discovered root and leaves the controller builds a P2MP policy and advertise it to the head-end router (i.e. the root of the P2MP Tree). In addition, the controller builds the replication segments on each segment of the tree, Root, Transit and Leaf nodes and downloads the forwarding instructions to the nodes.

As it was mentioned a SR p2mp policy is a variant of the SR policy and as such it reuses the concept of a candidate path. This draft reuses some of the concepts and TLVs mentioned in [I-D. draft-ietf-idr-segment-routing-te-policy]

A candidate path with in the P2MP policy can contain multiple path-
instances. A path-instance can be viewed as a P2MP LSP. For candidate path global optimization purposes two or more path-instances can be used to execute make before break procedures.

Each path-instance is a P2MP LSP as such each path-instance needs a set of replication segments to construct its forwarding instructions.

2. Conventions used in this document

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

3. Design of the Data Model
submodule: router-p2mp-traffic-engineering (belongs-to root)
---rw p2mp-traffic-engineering!
  +--rw p2mp-policy* [root-address tree-id]
    +--rw root-address     inet:ip-address
    +--rw tree-id         uint32
    +--rw p2mp-policy-name? string
    +--rw admin-state?    enumeration
    +--ro oper-state?     enumeration
    +--rw leaf-list* [leaf-address]
    |  +--rw leaf-address     inet:ip-address
    |  +--rw admin-state?    enumeration
    +--rw candidate-path* [protocol-id originator discriminator]
    |  +--rw protocol-id     enumeration
    |  +--rw originator      inet:ip-address
    |  +--rw discriminator   uint32
    |  +--rw candidate-path-name? string
    |  +--rw admin-state?    enumeration
    |  +--ro oper-state?     enumeration
    |  +--rw preference?     uint32
    |  +--rw constraints* [index]
    |     |  +--rw index        uint32
    |     |  +--rw attributes?  uint32
    |  +--rw explicit-routing* [index]
    |     |  +--rw index        uint32
    |     |  +--rw attributes?  uint32
    |  +--rw path-instances* [index]
    |     |  +--rw index        uint32
    |     |  +--rw attributes?  uint32
    +--rw replication-segment* [node-address replication-id]
    +--rw node-address      inet:ipv4-address
    +--rw replication-id    uint32
    +--rw admin-state?     enumeration
    +--ro oper-state?      enumeration
    +--rw root-address?    inet:ipv4-address
    +--rw tree-id?         uint32
    +--rw instance-id?     uint32
    +--rw replication-sid? uint32
    +--rw downstream-nodes* [downstream-index]
    |  +--rw downstream-index uint32
    |  +--rw next-hop-address?     inet:ip-address
    |  +--rw next-hop-interface-name? if:interface-ref
    |  +--rw protecting-next-hop?   boolean
    |  +--rw protect-nexthop-id?    uint32
    +--rw (label)?
      +--:(sid-list)
      |  +--rw sid-list* [index]
4. Configuration

This Module augments the "/rt:routing:" with a treeSID container. This container defines all the configuration parameter related to treeSID and P2MP SR Policy for this particular routing. The P2MP SR policy contains replication policy which in order contain candidate-path and the next-hop-groups for each OIF in the replication Policy. IT should be noted that two disjoint replication policies can be connected via a SR Policy as per draft-ietf-spring-segment-routing-policy.

5. Control plane configuration

6. States

7. Yang Data Model
<CODE BEGINS> file "ietf-p2mp-policy-@2019-07-04.yang"
module ietf-tree-sid {
    yang-version 1.1;
    namespace "urn:ietf:params:xml:ns:yang:p2mp-policy-segment";
    // replace with IANA namespace when assigned
    prefix tree-sid;

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

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

    import ietf-routing-types {
        prefix "rt-types";
    }

    import ietf-routing {
        prefix "rt";
    }

    import ietf-interfaces {
        prefix "if";
    }

    import ietf-ip {
        prefix ip;
    }

    organization
        "IETF SPRING Working Group";

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

                WG Chair: Bruno Decraene
                <mailto:bruno.decreaene@orange.com>

                WG Chair: Rob Shakir
                <mailto:robs@google.com>

                Editor: Hooman Bidgoli
                <mailto:hooman.bidgoli@nokia.com>

                Editor: Tanmoy Kundu
                <mailto:tanmoy.kundu@nokia.com>
The module defines a collection of YANG definition for p2mp policy module.

First draft.

RFC XXXX: A YANG Data Model for TREE-SID

Create p2mp policies and their corresponding candidate paths for p2mp trees.

Each p2mp-policy identifies one or more p2mp LSP for on the root towards a set of leaf/leaves.

P2MP policy name to be referenced by mvpn pmsi.

Administratively enable/disable Tree SID p2mp policy.
```yang
leaf oper-state {
    type enumeration{
        enum down { value 0; }
        enum up   { value 1; }
    }
    default down;
    config false;
    description
        "Tree SID p2mp policy operational state based on users."
}

list leaf-list {
    key "leaf-address";
    uses leaf-list-key;
    description
        "This list consists of one or more endpoint/s for a p2mp-segment."
    leaf admin-state {
        type enumeration{
            enum down { value 0; }
            enum up   { value 1; }
        }
        default down;
        description
            "Administratively enable/disable each leaf/endpoint";
    }
}

list candidate-path {
    description
        "Candidate path is p2mp tree representing a unique path from root to a specific endpoint using the tree-id constraint."
    key "protocol-id originator descriminator";
    uses candidate-path-key;
    leaf candidate-path-name {
        type string;
        description
            "A candidate path name equivalent to a PLSP";
    }
}
```
leaf admin-state {
    type enumeration {
        enum down { value 0; }
        enum up { value 1; }
    }
    default down;
    description "Administratively enable/disable Tree SID candidate path."
}

leaf oper-state {
    type enumeration {
        enum down { value 0; }
        enum up { value 1; }
    }
    default down;
    config false;
    description "candidate-path operational state based on ilm programming."
}

leaf preference {
    type uint32;
    default 100;
    description "Preference determines the best preferred candidate-path among list of candidate path towards a leaf. Higher preference is chosen."
}

list constraints {
    description "Set of constraints";
    key "index";
    leaf index {
        type uint32;
        description "Key index"
    }
    leaf attributes {
        type uint32;
        description "Human to fill this, I am not sure";
    }
}
list explicit-routing {
  description
      "Set of explicit routing";
  key "index";
  leaf index {
      type uint32;
      description
          "Key index";
  } 
  leaf attributes {
      type uint32;
      description
           "Hooman to fill this, I am not sure";
  }
}

list path-instances {
  description
      "Set of calculated path
         given the constraints
         and explicit routing per
         candidate path";
  key "index";
  leaf index {
      type uint32;
      description
          "Key index";
  }
  leaf instance-id {
      type leafref {
      path "..//..//..//replication-segment/replication-id";
     }
  } 
  leaf oper-state {
      type enumeration{
        enum down  { value 0; }
        enum up    { value 1; }
      }
      default down;
      config false;
list replication-segment {
  description
  "A replication segment specifies the forwarding information for one path-instance in a candidate path.";
  key "node-address replication-id";
  uses replication-segment-key;
  leaf admin-state {
    type enumeration{
      enum down { value 0; }
      enum up { value 1; }
    }
    default down;
    description
    "Administratively enable/disable Tree SID replication segment.";
  }
  leaf oper-state {
    type enumeration{
      enum down { value 0; }
      enum up { value 1; }
    }
    default down;
    config false;
    description
    "Replication segment operational state based on SID programming.";
  }
  container service-info {
    leaf root-address {
      type inet:ipv4-address;
      description
    "Root address of the tree.";
    }
    leaf tree-id {

type uint32;
description
  "Tree ID uniquely identifies a tunnel in the root, 
  this also represent a specific constraint. Also known as 
  color and/or p2mp-id";
}
leaf instance-id {
  type uint32;
  description
    "Each LSP instance within a root, tree-id.";
}
leaf replication-sid {
  type uint32;
  description
    "incoming sid to identify this 
    replication segment. Either 
    BSID or MPLS label";
}
list downstream-nodes {
  description
    "Identifies each nexthop in a candidate path.";
  key "downstream-index";
  uses downstream-key;
  leaf next-hop-address {
    type inet:ip-address;
    description
      "Nexthop address of the destination.";
  }
  leaf next-hop-interface-name {
    type if:interface-ref;
    description
      "Next hop out going interface.";
  }
  leaf protecting-next-hop {
    type boolean;
    default false;
    description
      "true if this is a protect nexthop.";
  }
}
leaf protect-nexthop-id {
  type uint32;
  description  
    "Nexthop protection id.";
}

choice label {
  list sid-list {
    key index;
    uses sid-list-key;
    description  
      "Out going label for this nexthop.";
    leaf sid-segment-type {
      type uint32;
      description  
        "Hooman to fill, I don't know";
    }
  }
  list sr-policy {
    key replication-sid;
    uses sr-policy-key;
    description  
      "sr-policy sid for this nexthop.";
    leaf sr-policy {
      type string;
      description  
        "SR policy name to be referenced by outloing label";
    }
  }
  list rsvp-te {
    key replication-sid;
    uses sr-policy-key;
    description  
      "RSVP TE Tunnel for this nexthop.";
    leaf rsvp-te-tunnel-id {
      type uint32;
      description  
        "rsvp-te-tunnel-id";
    }
  }
}

// -------------------------- GROUPINGS ---------------------------
grouping p2mp-policy-key {
  leaf root-address {
    type inet:ip-address;
    description "Root address of the tree.";
  }
  leaf tree-id {
    type uint32;
    description "Tree ID uniquely identifies a tunnel in the root, 
    this also represents a specific constraint. Also known as color 
    and/or p2mp-id";
  }
}

grouping leaf-list-key {
  leaf leaf-address{
    type inet:ip-address;
    description "Leaf address of this p2mp-tree";
  }
}

grouping candidate-path-key {
  leaf protocol-id {
    type enumeration {
      enum pcep { value 10; } 
      enum bgp-sr-policy { value 20; } 
      enum configuration { value 30; } 
    } 
    description "Protocol-Origin of a candidate path is an 8-bit value 
    which identifies the component or protocol that originates 
    or signals the candidate path."
  }
  leaf originator {
    type inet:ip-address;
    description "128 bit value, IPv4 address are encoded in lower 32 bit."
  }
}
leaf descriminator {
    type uint32;
    description
    "The Discriminator is a 32 bit value associated
    with a candidate path that uniquely identifies
    it within the context of an SR Policy from a
    specific Protocol-Origin";
}

grouping replication-segment-key {
    leaf node-address {
        type inet:ipv4-address;
        description
        "Node address for which this replication policy is used.";
    }
    leaf replication-id {
        type uint32;
        description
        "Each LSP per candidate path.";
    }
}

grouping downstream-key {
    leaf downstream-index {
        type uint32;
        description
        "Nexthop group or downstream replication node index";
    }
}

grouping sid-list-key {
    leaf index {
        type uint32 {
            range 1..2;
        }
    }
}

grouping sr-policy-key {
    leaf replication-sid {
        type uint32 {
            range 1..12;
        }
        description
        "The Discriminator is a 32 bit value associated
        with a candidate path that uniquely identifies
        it within the context of an SR Policy from a
        specific Protocol-Origin";
    }
}
6. IANA Considerations

This document contains no actions for IANA.

7. Security Considerations

TBD

8. References

8.1. Normative References

8.2. Informative References


7. Acknowledgments

Authors’ Addresses

Hooman Bidgoli
Nokia
600 March Rd.
Ottawa, Ontario K2K 2E6
Canada

Email: hooman.bidgoli@nokia.com

Daniel Voyer
Bell Canada
Montreal
CA

Email: daniel.voyer@bell.ca