

SPRING Working Group
Internet-Draft
Intended status: Standards Track
Expires: October 9, 2021

K. Raza, Ed.
R. Sawaya
Cisco Systems

Z. Shunwan
Huawei Technologies

D. Voyer
Bell Canada

M. Durrani
Equinix

S. Matsushima
SoftBank

V. Beeram
Juniper Networks

April 7, 2021

YANG Data Model for Segment Routing Policy draft-ietf-spring-sr-policy-yang-01

Abstract

This document defines a YANG data model for Segment Routing (SR) Policy that can be used for configuring, instantiating, and managing SR policies. The model is generic and apply equally to the MPLS and SRv6 instantiations of SR policies.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of [BCP 78](#) and [BCP 79](#).

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

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

This Internet-Draft will expire on October 9, 2021.

Copyright Notice

Copyright (c) 2021 IETF Trust and the persons identified as the document authors. All rights reserved.

Raza, et al.

Expires October 9, 2021

[Page 1]

Internet-Draft

YANG Data Model for SR Policy

April 2021

This document is subject to [BCP 78](#) and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://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.	Specification of Requirements	3
3.	Building Blocks	3
4.	YANG Model	4
4.1.	Types and Definitions	5
4.2.	SR Policy	6
4.2.1.	Configuration	6
4.2.2.	State	10
4.2.3.	Notification	12
5.	Pending Items	13
6.	YANG Specification	14
6.1.	Types	14
6.2.	SR Policy	22
7.	Security Considerations	44
8.	IANA Considerations	44
9.	Acknowledgments	45
10.	References	45
10.1.	Normative References	45
10.2.	Informative References	46
	Authors' Addresses	46

[1.](#) Introduction

The Network Configuration Protocol (NETCONF) [[RFC6241](#)] defines

mechanisms to manage network devices. YANG [[RFC6020](#)] is a modular language that represents data structures in an XML tree format, and is used as a data modeling language for the NETCONF.

Segment Routing (SR), as defined in [[RFC8402](#)], allows a headend node to steer a packet flow along any topological path and/or service chain. The headend node is said to steer a flow into a Segment Routing Policy (SR Policy). An SR policy is a framework [[I-D.ietf-spring-segment-routing-policy](#)] that enables instantiation of an ordered list of segments on a node for implementing a policy.

This document introduces a YANG data model for SR policy framework for instantiating, configuring and managing SR policies along with its attributes. It is also expected that other companion models, such as BGP SR Policy [[I-D.ietf-idr-segment-routing-te-policy](#)], will be defined and/or augmented accordingly in their respective areas.

This model defines the following constructs for managing an SR policy:

- o Configuration
- o Operational State
- o Notifications
- o Executables (Actions)

This document expects and requires the reader to be well familiar with the concepts and constructs of an SR policy [[I-D.ietf-spring-segment-routing-policy](#)] as well as the YANG modeling language and its presentation [[RFC6020](#)].

[2.](#) Specification of Requirements

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 [BCP 14](#) [[RFC2119](#)] [[RFC8174](#)] when, and only when, they appear in all capitals, as shown here.

[3.](#) Building Blocks

Before looking into the YANG model for SR policy, it is important to recall and highlight the major building blocks and constructs that constitute and contribute to an SR policy, as described in [\[I-D.ietf-spring-segment-routing-policy\]](#).

- o **policy:** specifies constructs to allow a headend node to setup SR path(s) as an ordered list of segments for a given color and endpoint. The endpoint and the color are used to automate the steering of service or transport routes on an SR Policy. For a given headend, the key for an SR policy is (color, endpoint) where endpoint is an IP address that could be also NULL.
- o **candidate-path:** is the unit for signalling of an SR Policy to a headend via protocols (such as PCEP, BGP, CLI etc.). A candidate path is either dynamic or explicit type, where an explicit candidate path is associated with one or more segment-lists and

dynamic candidate path expresses optimization objectives and set of constraints. An SR Policy is associated with one or more candidate paths and the preference of the candidate path is used to select the best candidate path for an SR Policy. A candidate path is valid if it is usable (e.g. when its constituents SIDs are reachable). An "active" candidate path is the selected path (for forwarding) that is valid and determined to be the best path of the SR Policy. A candidate path is keyed by (protocol-origin, originator, discriminator).

- o **segment-list:** specifies ordered list of segments to traverse, where a segment can be specified in various forms (refer section 4 of [\[I-D.ietf-spring-segment-routing-policy\]](#)). The list is sorted by the index of the segment. A segment-list is used and referred by an explicit type of candidate-path. A segment-list is keyed by its name.
- o **binding-sid:** An SR policy is associated with a BSID to provide benefits of scaling, network opacity and service independence.

[4.](#) YANG Model

The modeling in this document complies with the Network Management Datastore Architecture (NMDA) [RFC8342]. The operational state data is combined with the associated configuration data in the same hierarchy [RFC8407]. When protocol states are retrieved from the NMDA operational state datastore, the returned states cover all "config true" (rw) and "config false" (ro) nodes defined in the schema.

For SR policy YANG specification, this document defines following new YANG modules:

Module Name	Purpose
ietf-sr-policy-types	defines common and basic types related to an SR policy and related constructs
ietf-sr-policy	defines the model for SR policy instantiation, configuration, and management

[4.1.](#) Types and Definitions

SR policy common types and definitions are defined in the new module "ietf-sr-policy-types". The main types defined in this module include:

- o dataplane-type: A union to specify MPLS or IPv6 as the dataplane type for SR.
- o sid-value-type: A Union to specify SID value for SR-MPLS or SRv6 type.
- o binding-sid-alloc-mode: Enum to define explicit or dynamic alloc mode types for a BSID.

- o protocol-origin-type: Enum to specify protocol origin (e.g. PCEP) for an SR policy.
- o explicit-binding-sid-rule-type: Enum to specify BSID alloc enforcement/rule when doing explicit alloc request.
- o binding-sid-oper-state: An Enum representing various operational states for a BSID.
- o policy-admin-state: An Enum for admin state of an SR policy.
- o policy-oper-state: An Enum for operational state of an SR policy.
- o segment-type: An Enum that defines various types for a "segment" of a Segment list.
- o candidate-path-non-selection-reason: The base identity along with its children to specify reason for not selecting a candidate path as the best/active path.
- o path-disjointness: The base identity for disjoint path computation. The disjointness types include link, node, srlg, srlg-node etc.
- o policy-down-reason: The base identity along with its children to specify reason for a policy becoming (or remaining) operationally down.
- o binding-sid-unavailable-reason: The base identity along with its children to specify reason for a BSID's unavailability.

The associated YANG specification for this module is captured in [Section 6.1](#).

[4.2](#). SR Policy

The base SR policy model is captured in ietf-sr-policy module. This base module augments "/rt:routing" and specifies the configuration, operational state, executables/rpcs, and notification events required to manage SR policies.

The associated YANG specification for this module is captured in

[Section 6.2.](#)

[4.2.1.](#) Configuration

In terms of configuration hierarchy, SR policy configuration tree has following two main areas:

- o attributes: container that defines common constructs that could be used across policies. Examples of such a construct include segment-lists, affinity-map etc. In future revision of this document, it is expected that this container will have more constructs defined.
- o policies: container that defines list of policies with their attributes such as BSID, candidate-paths etc.

Following diagram depicts high level yang organization and hierarchy for an SR policy specification:

```

traffic-engineering
+ attributes
|   + affinity-map
|   |   ....
|   |
|   + segment-lists
|   |   segment-list* [name]
|   |       segments
|   |           segment* [index]
|   |           ...
|   + explicit-binding-sid-rules
|       ...
+ policies
  policy* [color endpoint]
  + ...
  |
  + binding-sid
  |   ...
  |
  + candidate-paths
    candidate-path* [protocol-origin originator discriminat
    + ...
    |
    + type
      + explicit
      |   segment-lists
      |       segment-list* [ref]
      |       ...
      + dynamic
      |   constraints
      |   ...

```

Figure 1: SR Policy - Hierarchy

Using the building blocks described in [Section 3](#), following is the complete graphical representation of the data model for SR policy configuration:

```

module: ietf-sr-policy
augment /rt:routing:
  +--rw segment-routing
  +--rw traffic-engineering
  +--rw attributes
  |   +--rw affinity-map

```



```

| | +--rw affinity* [name]
| | |   +--rw name          string
| | |   +--rw bit-position? uint16
| +--rw segment-lists
| | +--rw segment-list* [name]
| | |   +--rw name          string
| | |   +--rw segments
| | |     +--rw segment* [index]
| | |       +--rw index          uint32
| | |       +--rw type?          sr-policy-types:segment-type
| | |       +--rw segment-types
| | |         | +--rw segment-type-1
| | |         | |   +--rw sid-value?  rt-types:mpls-label
| | |         | +--rw segment-type-2
| | |         | |   +--rw sid-value?  srv6-types:srv6-sid
| | |         | +--rw segment-type-3
| | |         | |   +--rw ipv4-address?  inet:ipv4-address
| | |         | |   +--rw algorithm?      uint8
| | |         | +--rw segment-type-4
| | |         | |   +--rw ipv6-address?  inet:ipv6-address
| | |         | |   +--rw algorithm?      uint8
| | |         | +--rw segment-type-5
| | |         | |   +--rw ipv4-address?      inet:ipv4-address
| | |         | |   +--rw interface-identifier?  uint32
| | |         | +--rw segment-type-6
| | |         | |   +--rw local-ipv4-address?  inet:ipv4-address
| | |         | |   +--rw remote-ipv4-address?  inet:ipv4-address
| | |         | +--rw segment-type-7
| | |         | |   +--rw local-ipv6-address?      inet:ipv6-a
| | |         | |   +--rw local-interface-identifier?  uint32
| | |         | |   +--rw remote-ipv6-address?      inet:ipv6-a
| | |         | |   +--rw remote-interface-identifier?  uint32
| | |         | +--rw segment-type-8
| | |         | |   +--rw local-ipv6-address?      inet:ipv6-address
| | |         | |   +--rw remote-ipv6-address?      inet:ipv6-address
| | |         | +--rw segment-type-9
| | |         | |   +--rw ipv6-address?  inet:ipv6-address
| | |         | |   +--rw algorithm?      uint8
| | |         | +--rw segment-type-10
| | |         | |   +--rw local-ipv6-address?      inet:ipv6-a
| | |         | |   +--rw local-interface-identifier?  uint32
| | |         | |   +--rw remote-ipv6-address?      inet:ipv6-a
| | |         | |   +--rw remote-interface-identifier?  uint32
| | |         | +--rw segment-type-11
| | |         | |   +--rw local-ipv6-address?      inet:ipv6-address
| | |         | |   +--rw remote-ipv6-address?      inet:ipv6-address

```

```

| |             +---rw validate?          boolean
| +---rw explicit-binding-sid-rules* [index]

```

```

| +---rw index      uint32
| +---rw rule?      sr-policy-types:explicit-binding-sid-rule-type
+---rw policies
  +---rw policy* [color endpoint]
    +---rw color          uint32
    +---rw endpoint       inet:ip-address
    +---rw name?          string
    +---rw description?   string
    +---rw admin-state?   sr-policy-types:policy-admin-state
    +---rw priority?      uint8
    +---rw binding-sid
      | +---rw dataplane?   sr-policy-types:dataplane-type
      | +---rw value?       sr-policy-types:sid-value-type
    +---rw candidate-paths
      +---rw candidate-path* [protocol-origin originator discriminator]
        +---rw protocol-origin      sr-policy-types:protocol-origin
        +---rw originator            string
        +---rw discriminator         uint32
        +---rw preference             uint32
        +---rw name?                 string
        +---rw description?          string
        +---rw binding-sid {capability-candidate-path-binding-sid}
          | +---rw dataplane?   sr-policy-types:dataplane-type
          | +---rw value?       sr-policy-types:sid-value-type
        +---rw (type)?
          +---:(explicit)
            | +---rw segment-lists
            |   +---rw segment-list* [name-ref]
            |   +---rw name-ref      -> /rt:routing/sr-policy:segment-list
            |   +---rw weight?       uint32
          +---:(dynamic)
            +---rw sid-dataplane-type?      sr-policy-types:dataplane-type
            +---rw constraints
              +---rw affinities
                | +---rw exclude-any*   string
                | +---rw include-any*    string
                | +---rw include-all*   string
              +---rw bounds
                | +---rw igp-metric-bound?      uint32

```

```

|   +---rw te-metric-bound?      uint32
|   +---rw latency-metric-bound? uint32
|   +---rw segment-bound?       uint32
+---rw segment-rules
|   +---rw sid-algorithm?      uint8
+---rw disjoint-path
|   +---rw group-id?           uint32
|   +---rw disjointness-type?  identityref
|   +---rw subgroup-id?       uint32

```

Figure 2: SR Policy - Config Tree

Please take note of the following important points in the above configuration model:

- o This model supports both MPLS and SRv6 dataplane for SR -- i.e. items like segments and BSID can be defined as MPLS label or SRv6 SIDs.
- o Specification of a segment supports all the types defined in SR policy base specification document
- o The above model supports explicit BSID specification on SR policy level as the main mode of specification. The model also allows explicit BSID per candidate-path as an if-feature capability that is optional for implementations
- o The above model will be extended in future revisions of this document to enhance constraints specification for dynamic type of candidate-path, as well as add traffic-steering controls.

[4.2.2.](#) State

As per NMDA model, the state related to configuration items specified in earlier [Section 4.2.1](#) can be retrieved from the same tree. This section defines the other operational state items related to SR policy.

In addition to configured state, the operational state corresponding to the SR policy includes:

- o policy operational state

- o policy up/down timestamps
- o policy BSID info such as alloc mode, actual value in-use, operational state, and forwarding stats
- o Per candidate-path info such as:
 - * Whether candidate-path is the best candidate-path
 - * In case of non-best, the reason for such non-selection
 - * Type of candidate-path - explicit or dynamic
 - * Per segment-list information - such as validity of the segment-list, as well as forwarding state for a valid segment-list.

The forwarding state is represented in terms of per forwarding path info that includes nexthop address, outgoing interface, protection information, and encapsulation (label stack or SRv6 SID stack) etc.

Following is a simplified graphical representation of the data model for the SR policy (derived) operational state:

```

module: ietf-sr-policy
  augment /rt:routing:
    +--rw segment-routing
      +--rw traffic-engineering
        +--rw policies
          +--rw policy* [color endpoint]
            +--rw color                uint32
            +--rw endpoint              inet:ip-address
            +--ro oper-state?           sr-policy-types:policy-oper-state
            +--ro transition-count?     uint32
            +--ro up-time?              yang:date-and-time
            +--ro down-time?            yang:date-and-time
            +--rw binding-sid
              | +--ro alloc-mode?       sr-policy-types:binding-sid-alloc-mod
              | +--ro allocated-sid?    sr-policy-types:sid-value-type
              | +--ro oper-state?       sr-policy-types:binding-sid-oper-stat

```

```

+--ro counters
|   +--ro pkts?      yang:counter64
|   +--ro octets?    yang:counter64
+--rw candidate-paths
    +--rw candidate-path* [protocol-origin originator discriminator]
        +--rw protocol-origin      sr-policy-types:protocol-origin
        +--rw originator            string
        +--rw discriminator          uint32
        +--rw name                    string
        +--ro is-best-candidate-path? boolean
        +--ro non-selection-reason?  identityref
        +--ro is-valid?              boolean
        +--ro forwarding-paths
            +--ro forwarding-path* [path-id]
                +--ro path-id          uint8
                +--ro next-hop-address? inet:ip-address
                +--ro next-hop-table-id? uint32
                +--ro interface?       if:interface-ref
                +--ro sid-list
                    |   +--ro (dataplanetype)?
                    |       +--:(mpls)
                    |       |   +--ro labels* [label]
                    |       |       +--ro label    rt-types:mpls-label

```

```

|       +--:(srv6)
|       +--ro sids* [sid]
|       +--ro sid    srv6-types:srv6-sid
+--ro is-protected?    boolean
+--ro is-pure-backup?  boolean
+--ro backup-path-id?  uint8
+--ro weight?          uint32

```

Figure 3: SR Policy - State Tree

[4.2.3.](#) Notification

This model defines a list of notifications to inform an operator of important events detected regarding an SR policy. These events include events related to:

- o policy status: policy operational state changes

- o Candidate-path active status and changes
- o Explicit Binding SID collision/unavailability events

Following is a simplified graphical representation of the data model for SR policy notifications:

module: ietf-sr-policy

notifications:

```

+---n sr-policy-oper-state-change-event
|   +---ro policy-name-ref?          -> /rt:routing/sr-policy:segment-routing/
|   +---ro policy-color-ref?         -> /rt:routing/sr-policy:segment-routing/
|   +---ro policy-endpoint-ref?      -> /rt:routing/sr-policy:segment-routing/
|   +---ro policy-new-oper-state?    sr-policy-types:policy-oper-state
|   +---ro policy-down-reason?       identityref
+---n sr-policy-candidate-path-change-event
|   +---ro policy-name-ref?          -> /rt:routing/sr-policy:segment-routing/tr

```

```

|   +---ro policy-color-ref?      -> /rt:routing/sr-policy:segment-routing/tr
|   +---ro policy-endpoint-ref?   -> /rt:routing/sr-policy:segment-routing/tr
|   +---ro existing-preference?   uint32
|   +---ro new-preference?        uint32
+---n sr-policy-binding-sid-unavailable-event
|   +---ro policy-name-ref?       -> /rt:routing/sr-policy:segment-r
|   +---ro policy-color-ref?      -> /rt:routing/sr-policy:segment-r
|   +---ro policy-endpoint-ref?   -> /rt:routing/sr-policy:segment-r
|   +---ro policy-binding-sid-value-ref? -> /rt:routing/sr-policy:segment-r
|   +---ro reason?                identityref
+---n sr-policy-candidate-path-binding-sid-mismatch-event
    +---ro policy-color-ref?      -> /rt:routin
    +---ro policy-endpoint-ref?   -> /rt:routin
    +---ro existing-candidate-path-protocol-origin-ref? -> /rt:routin
    +---ro existing-candidate-path-preference-ref?     -> /rt:routin
    +---ro existing-candidate-path-binding-sid-dataplane-ref? -> /rt:routin
    +---ro existing-candidate-path-binding-sid-value-ref? -> /rt:routin
    +---ro conflicting-candidate-path-protocol-origin? uint8
    +---ro conflicting-candidate-path-preference?      uint32
    +---ro conflicting-candidate-path-binding-sid-dataplane? sr-policy-typ
    +---ro conflicting-candidate-path-binding-sid-value?  sr-policy-typ

```

Figure 4: SR Policy - Notification Tree

5. Pending Items

Following are the items that will be addressed in future revisions of this document:

- o Configuration and Specification of:
 - * Traffic steering over SR policy
 - * ODN templates
 - * Spray policy

- o Executables (RPC actions)

6. YANG Specification

Following are actual YANG definition for the modules defined earlier in the document.

[6.1.](#) Types

```
<CODE BEGINS> file "ietf-sr-policy-types@2019-11-04.yang"

module ietf-sr-policy-types {
  namespace "urn:ietf:params:xml:ns:yang:ietf-sr-policy-types";

  prefix "sr-policy-types";

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

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

  import ietf-srv6-types {
    prefix "srv6-types";
  }

  organization "IETF SPRING Working Group";

  contact
    "WG Web:   <http://tools.ietf.org/wg/spring/>
    WG List:   <mailto:spring@ietf.org>
    Editor:    Kamran Raza
               <mailto:skraza@cisco.com>
    Editor:    Zhuang Shunwan
               <mailto:zhuangshunwa@huawei.com>
    Editor:    Daniel Voyer
               <mailto:daniel.voyer@bell.ca>
    Editor:    Muhammad Durrani
               <mailto:mdurrani@equinix.com>
    Editor:    Satoru Matsushima
               <mailto:satoru.matsushima@g.softbank.co.jp>
    Editor:    Pavan Vishnu Beeram
               <mailto:vbeeram@juniper.net>
    ";
```



```
description
  "This YANG module defines the essential types for the management
  of SR policy module.
  Copyright (c) 2019 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 "2019-11-04" {
  description
    "New editor added";
  reference
    "draft-raza-spring-sr-policy-yang";
}

revision "2019-07-08" {
  description
    "Dynamic TE candidate-path support";
  reference
    "draft-raza-spring-sr-policy-yang";
}

revision "2018-07-01" {
  description
    "Initial version";
  reference
    "draft-raza-spring-sr-policy-yang";
}

/* Identities */
identity candidate-path-not-selected-reason {
  description
    "Base identity for which reasons for not selecting
    candidate path are derived from";
}
identity candidate-path-not-selected-not-best {
  base candidate-path-not-selected-reason;
  description
    "Higher preference path exists";
}
identity candidate-path-not-selected-no-valid-segment-list {
  base candidate-path-not-selected-reason;
  description
```

"Candidate path has no valid segment list(s)";

```
}
identity candidate-path-not-selected-empty-segment-list {
  base candidate-path-not-selected-reason;
  description
    "Candidate path has empty segment list(s)";
}
identity candidate-path-not-selected-invalid-binding-sid {
  base candidate-path-not-selected-reason;
  description
    "Candidate path has invalid binding SID";
}

identity policy-down-reason {
  description
    "Base identity for the reasons why SR policy is operationally down";
}
identity policy-down-reason-admin-down {
  base policy-down-reason;
  description "Policy is administrately down";
}
identity policy-down-reason-no-source-address {
  base policy-down-reason;
  description "Policy has no source address";
}
identity policy-down-reason-no-endpoint {
  base policy-down-reason;
  description "Policy has no end-point";
}
identity policy-down-reason-no-candidate-path {
  base policy-down-reason;
  description "Policy has no candidate path";
}
identity policy-down-reason-no-valid-candidate-path {
  base policy-down-reason;
  description "Policy has no valid candidate path";
}
identity policy-down-reason-candidate-path-invalid-segment-list {
  base policy-down-reason;
  description "Policy's candidate path has invalid segment list";
}
```

```

identity policy-down-reason-policy-unconfigured {
    base policy-down-reason;
    description "Policy is unconfigured";
}
identity policy-down-reason-policy-color-endpoint-updated {
    base policy-down-reason;
    description "Policy's color and end-point are updated";
}

```

```

identity policy-down-reason-local-label-setup-failed {
    base policy-down-reason;
    description "Policy's local label setup (allocation/rewrite) failed";
}
identity policy-down-reason-forwarding-rewrite-failed {
    base policy-down-reason;
    description "Policy's forwarding rewrite installation failed";
}
identity policy-down-reason-internal-error {
    base policy-down-reason;
    description "Infra related internal error";
}

identity binding-sid-unavailable-reason {
    description
        "Base identity for binding sid unavailable reason types";
}
identity binding-sid-allocation-error {
    base binding-sid-unavailable-reason;
    description "SID allocator returned an error";
}
identity binding-sid-already-exists {
    base binding-sid-unavailable-reason;
    description "Binding sid already exists/allocated";
}
identity binding-sid-internal-error {
    base binding-sid-unavailable-reason;
    description "Internal error with binding sid allocation";
}
identity binding-sid-color-endpoint-conflict {
    base binding-sid-unavailable-reason;
    description "Binding sid already allocated by another sr-policy with differ
}

```

```

identity binding-sid-rewrite-error {
    base binding-sid-unavailable-reason;
    description "Binding sid forwarding rewrite error";
}
identity binding-sid-outside-srlb-range {
    base binding-sid-unavailable-reason;
    description "Binding sid outside SRLB range";
}

identity path-disjointness {
    description
        "Base identity for the type of path disjointness computation";
}
identity path-disjointness-link {
    base path-disjointness;

```

```

    description "The computed path is link-disjoint with the existing path";
}
identity path-disjointness-node {
    base path-disjointness;
    description "The computed path is node-disjoint with the existing path";
}
identity path-disjointness-srlg {
    base path-disjointness;
    description "The computed path is srlg-disjoint with the existing path";
}
identity path-disjointness-srlg-node {
    base path-disjointness;
    description "The computed path is node and srlg disjoint with the existing
}

/* Typedefs */
typedef sid-value-type {
    type union {
        type rt-types:mpls-label;
        type srv6-types:srv6-sid;
    }
    description "The SID value type";
}

typedef binding-sid-oper-state {
    type enumeration {

```

```

enum ALLOC-PENDING {
    value 1;
    description "SID allocation pending for Binding SID";
}
enum PROGRAMMED {
    value 3;
    description "Binding SID is programmed in forwarding";
}
enum CONFLICT {
    value 4;
    description "Binding SID is in-conflict state with
        regards to SID allocation. This also means that SID
        allocation is pending";
}
}
description
    "Binding SID operational state type";
}

typedef policy-admin-state {
    type enumeration {
        enum UP {

```

```

        value 1;
        description "SR policy is administratively up";
    }
    enum DOWN {
        value 2;
        description "SR policy is administratively down";
    }
}
description "SR policy admin state";
}

typedef policy-oper-state {
    type enumeration {
        enum UP {
            value 1;
            description "SR policy is operationally up";
        }
        enum DOWN {
            value 2;

```

```

        description "SR policy is operationally down";
    }
}
description "SR policy oper state";
}

typedef segment-type {
    type enumeration {
        enum segment-type-1 {
            value 1;
            description "SR-MPLS Label";
        }
        enum segment-type-2 {
            value 2;
            description "SRv6 SID";
        }
        enum segment-type-3 {
            value 3;
            description "IPv4 Prefix with optional SR Algorithm";
        }
        enum segment-type-4 {
            value 4;
            description "IPv6 Global Prefix with optional SR Algorithm for SR-MPLS";
        }
        enum segment-type-5 {
            value 5;
            description "IPv4 Prefix with Local Interface ID";
        }
        enum segment-type-6 {

```

```

        value 6;
        description "IPv4 Addresses for link endpoints as Local, Remote pair";
    }
    enum segment-type-7 {
        value 7;
        description "IPv6 Prefix and Interface ID for link endpoints as Local,
            Remote pair for SR-MPLS";
    }
    enum segment-type-8 {
        value 8;
        description "IPv6 Addresses for link endpoints as Local, Remote pair fo
            SR-MPLS";

```

```

    }
    enum segment-type-9 {
        value 9;
        description "IPv6 Global Prefix with optional SR Algorithm for SRv6";
    }
    enum segment-type-10 {
        value 10;
        description "IPv6 Prefix and Interface ID for link endpoints as Local,
            Remote pair for SRv6";
    }
    enum segment-type-11 {
        value 11;
        description "IPv6 Addresses for link endpoints as Local, Remote pair fo
            SRv6";
    }
}
description "SR segment type";
}

typedef dataplane-type {
    type enumeration {
        enum mpls {
            value 1;
            description "Segment-routing MPLS";
        }
        enum srv6 {
            value 2;
            description "Segment-routing v6";
        }
    }
    description "Dataplane type of the segments";
}

typedef binding-sid-alloc-mode {
    type enumeration {
        enum explicit {

```

```

        value 1;
        description "Explicitly specified BSID";
    }
    enum dynamic {
        value 2;

```

```

        description "Dynamically allocated BSID";
    }
}
description "binding SID allocation mode";
}

typedef protocol-origin-type {
    type enumeration {
        enum pcep {
            value 10;
            description "PCEP used as signalling mechanism for the candidate path";
        }
        enum bgp {
            value 20;
            description "BGP used as signalling mechanism for the candidate path";
        }
        enum local {
            value 30;
            description "CLI, Yang model via Netconf, gRPC, etc used for candidate";
        }
    }

    description "Originating protocol type";
}

typedef explicit-binding-sid-rule-type {
    type enumeration {
        enum enforce-srlb {
            value 1;
            description
                "Explicit Binding SID is enforced with no
                 fallback if label does not fall in SRLB or
                 if no SRLB is configured";
        }
        enum fallback-dynamic {
            value 2;
            description
                "Explicit Binding SID falls back to dynamic in
                 case explicit label is not available.";
        }
    }
    description "Explicit binding sid rule types";
}

```



```
} // module
```

```
<CODE ENDS>
```

Figure 5: ietf-sr-policy-types.yang

[6.2.](#) SR Policy

```
<CODE BEGINS> file "ietf-sr-policy@2019-11-04.yang"
```

```
module ietf-sr-policy {  
  
    namespace "urn:ietf:params:xml:ns:yang:ietf-sr-policy";  
  
    prefix "sr-policy";  
  
    import ietf-inet-types {  
        prefix "inet";  
    }  
  
    import ietf-interfaces {  
        prefix if;  
    }  
  
    import ietf-routing {  
        prefix "rt";  
    }  
  
    import ietf-routing-types {  
        prefix "rt-types";  
    }  
  
    import ietf-yang-types {  
        prefix "yang";  
    }  
  
    import ietf-srv6-types {  
        prefix "srv6-types";  
    }  
  
    import ietf-sr-policy-types {  
        prefix "sr-policy-types";  
    }  
  
    organization "IETF SPRING Working Group";  
}
```

Internet-Draft

YANG Data Model for SR Policy

April 2021

contact

"WG Web: <<http://tools.ietf.org/wg/spring/>>

WG List: <<mailto:spring@ietf.org>>

Editor: Kamran Raza
<<mailto:skraza@cisco.com>>

Editor: Zhuang Shunwan
<<mailto:zhuangshunwa@huawei.com>>

Editor: Daniel Voyer
<<mailto:daniel.voyer@bell.ca>>

Editor: Muhammad Durrani
<<mailto:mdurrani@equinix.com>>

Editor: Satoru Matsushima
<<mailto:satoru.matsushima@g.softbank.co.jp>>

Editor: Pavan Vishnu Beeram
<<mailto:vbeeram@juniper.net>>

";

description

"This module contains a collection of YANG definitions
for SR policy module.

Copyright (c) 2019 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 "2019-11-04" {

description

"Changes in keys for policy and its candidate paths";

reference

```

    "draft-raza-spring-sr-policy-yang";
}

revision "2019-07-08" {
    description
        "Dynamic TE candidate-path support";

```

```

    reference
        "draft-raza-spring-sr-policy-yang";
}

revision "2018-07-01" {
    description
        "Initial version";
    reference
        "draft-raza-spring-sr-policy-yang";
}

grouping segment_config {
    description "Segment properties grouping";
    leaf index {
        type uint32;
        description "Segment index";
    }
    leaf type {
        type sr-policy-types:segment-type;
        description "Segment type";
    }
}

container segment-types {
    description "Types of segments";
    container segment-type-1 {
        description
            "Segment declared by MPLS label";
        leaf sid-value {
            type rt-types:mpls-label;
            description "MPLS label value";
        }
    }
}

container segment-type-2 {
    description
        "Segment declared by SRv6 SID value";
    leaf sid-value {

```

```

        type srv6-types:srv6-sid;
        description "SRv6 SID value";
    }
}
container segment-type-3 {
    description
        "Segment declared by IPv4 Prefix with optional SR Algorithm";
    leaf ipv4-address {
        type inet:ipv4-address;
        description "Segment IPv4 address";
    }
    leaf algorithm {
        type uint8;
    }
}

```

```

        description "Prefix SID algorithm identifier";
    }
}
container segment-type-4 {
    description
        "Segment declared by IPv6 Global Prefix with optional
        SR Algorithm for SR-MPLS";
    leaf ipv6-address {
        type inet:ipv6-address;
        description "Segment IPv6 address";
    }
    leaf algorithm {
        type uint8;
        description "Prefix SID algorithm identifier";
    }
}
container segment-type-5 {
    description
        "Segment declared by IPv4 Prefix with Local Interface ID";
    leaf ipv4-address {
        type inet:ipv4-address;
        description "Node IPv4 address";
    }
    leaf interface-identifier {
        type uint32;
        description "local interface identifier";
    }
}
}

```

```

container segment-type-6 {
  description
    "Segment declared by IPv4 Addresses for link endpoints
    as Local, Remote pair";
  leaf local-ipv4-address {
    type inet:ipv4-address;
    description "Segment local IPv4 adjacency address";
  }
  leaf remote-ipv4-address {
    type inet:ipv4-address;
    description "Segment remote IPv4 adjacency address";
  }
}
container segment-type-7 {
  description
    "Segment declared by IPv6 Prefix and Interface ID for
    link endpoints as Local, Remote pair for SR-MPLS";
  leaf local-ipv6-address {
    type inet:ipv6-address;
    description "Local link IPv6 address";
  }

```

```

}
leaf local-interface-identifier {
  type uint32;
  description "Local interface identifier";
}
leaf remote-ipv6-address {
  type inet:ipv6-address;
  description "Remote link IPv6 address";
}
leaf remote-interface-identifier {
  type uint32;
  description "Remote interface identifier";
}
}
container segment-type-8 {
  description
    "Segment declared by IPv6 Addresses for link endpoints as
    Local, Remote pair for SR-MPLS";
  leaf local-ipv6-address {
    type inet:ipv6-address;
    description "Segment local IPv6 adjacency address";
  }

```

```

    }
    leaf remote-ipv6-address {
        type inet:ipv6-address;
        description "Segment remote IPv6 adjacency address";
    }
}
container segment-type-9 {
    description
        "Segment declared by IPv6 Global Prefix with optional
        SR Algorithm for SRv6";
    leaf ipv6-address {
        type inet:ipv6-address;
        description "Segment IPv6 prefix";
    }
    leaf algorithm {
        type uint8;
        description "Prefix SID algorithm identifier";
    }
}
container segment-type-10 {
    description
        "Segment declared by IPv6 Prefix and Interface ID for
        link endpoints as Local, Remote pair for SRv6";
    leaf local-ipv6-address {
        type inet:ipv6-address;
        description "Local link IPv6 address";
    }
}

```

```

    leaf local-interface-identifier {
        type uint32;
        description "Local interface identifier";
    }
    leaf remote-ipv6-address {
        type inet:ipv6-address;
        description "Remote link IPv6 address";
    }
    leaf remote-interface-identifier {
        type uint32;
        description "Remote interface identifier";
    }
}
container segment-type-11 {

```

```

    description
      "Segment declared by IPv6 Addresses for link endpoints as
      Local, Remote pair for SRv6";
    leaf local-ipv6-address {
      type inet:ipv6-address;
      description "Segment local IPv6 adjacency address";
    }
    leaf remote-ipv6-address {
      type inet:ipv6-address;
      description "Segment remote IPv6 adjacency address";
    }
  }
}
leaf validate {
  type boolean;
  default 'false';
  description "Indicates whether the segment should be validated. The default
  applies to all segments other than the first segment. For the
  first segment, validation is always done.";
}
}

grouping segment-properties {
  description
    "SR segment properties grouping";
  uses segment_config;
}

grouping attributes {
  description
    "Grouping containing attributes applicable to all SR policies";

  container attributes {
    description

```

```

    "Attributes applicable to SR policies";

    uses affinity-mapping;
    uses segment-lists;
    uses explicit-binding-sid-rules;
  }
}

```

```

grouping segment-lists {
  description
    "Segment lists grouping";
  container segment-lists {
    description "Segment-lists properties";

    list segment-list {
      key "name";
      description "Segment-list properties";
      leaf name {
        type string;
        description "Segment-list name";
      }
      container segments {
        description
          "Segments for given segment list";

        list segment {
          key "index";
          description "Configure Segment/hop at the index";
          uses segment-properties;
        }
      }
    }
  }
}

grouping binding-sid_config {
  description
    "Binding SID configuration properties grouping";
  leaf dataplane {
    type sr-policy-types:dataplane-type;
    description "Binding SID dataplane type";
  }
  leaf value {
    type sr-policy-types:sid-value-type;
    description "Binding SID value";
  }
}

```



```

grouping forwarding-counters {
  description
    "Grouping for counters";
  container counters {
    config false;
    description
      "Counters containing stats related to forwarding";

    leaf pkts {
      type yang:counter64;
      description "Number of packets forwarded";
    }
    leaf octets {
      type yang:counter64;
      units "byte";
      description "Number of bytes forwarded";
    }
  }
}

grouping binding-sid_state {
  description
    "Binding SID state properties grouping";
  leaf alloc-mode {
    type sr-policy-types:binding-sid-alloc-mode;
    config false;
    description "Binding SID type";
  }
  leaf allocated-sid {
    type sr-policy-types:sid-value-type;
    config false;
    description "Allocated SID value for the Binding SID";
  }
  leaf oper-state {
    type sr-policy-types:binding-sid-oper-state;
    config false;
    description
      "Binding SID operational state";
  }
}

grouping binding-sid-properties {
  description
    "Binding SID properties grouping";
  container binding-sid {
    description "Binding Segment ID";
    uses binding-sid_config;
    uses binding-sid_state;
  }
}

```

Internet-Draft

YANG Data Model for SR Policy

April 2021

```
    }  
  }  
  
  grouping mpls-label-stack {  
    description  
      "Grouping for MPLS label stack";  
  
    list labels {  
      key "label";  
      description  
        "Stack containing MPLS labels";  
  
      leaf label {  
        type rt-types:mpls-label;  
        description  
          "MPLS label value";  
      }  
    }  
  }  
  
  grouping srv6-sid-stack {  
    description  
      "Grouping for SRv6 label stack";  
  
    list sids {  
      key "sid";  
      description  
        "Stack containing SRv6 SIDs";  
  
      leaf sid {  
        type srv6-types:srv6-sid;  
        description  
          "SRv6 sid value";  
      }  
    }  
  }  
  
  grouping path-forwarding_state {  
    description "Policy Forwarding path information";  
    leaf path-id {  
      type uint8;  
      description "Primary path id";  
    }  
  }
```

```

leaf next-hop-address {
    type inet:ip-address;
    description "Nexthop address";
}
leaf next-hop-table-id {

```

```

    type uint32;
    description "Table ID for nexthop address";
}
leaf interface {
    type if:interface-ref;
    description "Outgoing interface handle";
}
container sid-list {
    description
        "Outgoing sid stack";
    choice dataplanetype {
        description
            "Outgoing sids dataplane choice";
        case mpls {
            uses mpls-label-stack;
        }
        case srv6 {
            uses srv6-sid-stack;
        }
    }
}
leaf is-protected {
    type boolean;
    description "Is this path protected ?";
}
leaf is-pure-backup {
    type boolean;
    description "Is this path a pure backup ?";
}
leaf backup-path-id {
    type uint8;
    description "Backup path id";
}
leaf weight {
    type uint32;
    description "Path's weight for W-ECMP balancing";
}

```

```

    }
}

grouping cpath-cmn-properties {
    description
        "Common properties of the candidate path";

    leaf is-valid {
        type boolean;
        config false;
        description
            "True if the segment-list is valid, False otherwise";
    }
}

```

```

    }

    container forwarding-paths {
        config false;
        description
            "Forwarding state of paths";
        list forwarding-path {
            key "path-id";
            description "Forwarding path";
            uses path-forwarding_state;
        }
    }
}

grouping explicit-path-properties {
    description
        "Explicit path properties of the candidate path";
    container segment-lists {
        description
            "Path segment list(s) properties";
        list segment-list {
            key "name-ref";
            description "SR policy candidate path segment lists";

            leaf name-ref {
                type leafref {
                    path "/rt:routing/sr-policy:segment-routing/sr-policy:traffic-engine";
                }
                description "Reference to segment-list name";
            }
        }
    }
}

```

```

    }
    leaf weight {
        type uint32;
        description "Segment-list weighted loadshare";
    }
}
}
}

grouping affinity-mapping {
    description "Affinity-map grouping";

    container affinity-map {
        description
            "Mapping of affinity names to bit position";
        list affinity {
            key "name";
            unique "bit-position";
            leaf name {

```

```

        type string;
        description
            "Name of the affinity";
    }
    leaf bit-position {
        type uint16;
        description
            "The affinity entry in this list is mapped to the this bit-position
            affinity bitmap";
    }

    description "Affinity";
}
}
}

grouping dynamic-path-properties {
    description
        "Dynamic path properties of the candidate path";
    leaf sid-dataplane-type {
        type sr-policy-types:dataplane-type;
        description
            "The dataplane type for the sid";

```

```

}

container constraints {
  description "Constraints for the dynamic path computation";
  container affinities {
    description "Affinity constraints on the computed dynamic path";
    leaf-list exclude-any {
      type string;
      description
        "The link is excluded if it has any of these affinities.";
    }
    leaf-list include-any {
      type string;
      description
        "The link is accepted if it has any of these affinities";
    }
    leaf-list include-all {
      type string;
      description
        "The link is accepted if it has all these affinities";
    }
  }
}

container bounds {
  description "Upper-bound constraints on the computed dynamic path";
  leaf igp-metric-bound {

```

```

    type uint32;
    description
      "Path is invalid if its IGP metric exceeds this value";
  }
  leaf te-metric-bound {
    type uint32;
    description
      "Path is invalid if its TE metric exceeds this value";
  }
  leaf latency-metric-bound {
    type uint32;
    units "microsecond";
    description
      "Path is invalid if its latency exceeds this value";
  }
}

```

```

    leaf segment-bound {
      type uint32;
      description
        "Path is invalid if it has more segments than this value";
    }
  }
  container segment-rules {
    description "Constraints on the segments to be used in the path";
    leaf sid-algorithm {
      type uint8 {
        range "128..255";
      }
      description
        "The prefix-sid algorithm to be used in path calculation";
    }
  }
  container disjoint-path {
    description "Path disjointness constraints";
    leaf group-id {
      type uint32 { range "1..65535"; }
      description "";
    }
    leaf disjointness-type {
      type identityref { base sr-policy-types:path-disjointness; }
      description
        "Type of disjointness computation used to find the path";
    }
    leaf subgroup-id {
      type uint32 { range "1..65535"; }
      description "";
    }
  }
}

```

```

}

```

```

grouping candidate-path_state {
  description
    "Candidate path state properties grouping";
  leaf is-best-candidate-path {
    type boolean;
    default 'false';
  }
}

```

```

        config false;
        description
            "True if the candidate path is the best candidate path, False otherwise";
    }
    leaf non-selection-reason {
        type identityref {
            base sr-policy-types:candidate-path-not-selected-reason;
        }
        config false;
        description
            "Candidate path not selected reason";
    }
}

grouping policy-properties_config {
    description
        "SR policy configuration grouping";
    leaf name {
        type string {
            length "1..59";
        }
        description "SR policy name";
    }
    leaf color {
        type uint32 {
            range "1..4294967295";
        }
        description "Color associated with the policy";
    }
    leaf endpoint {
        type inet:ip-address;
        description "Policy end point IP address";
    }
    leaf description {
        type string;
        description "Description of the policy";
    }
    leaf admin-state {
        type sr-policy-types:policy-admin-state;
        default 'UP';
    }
}

```

description


```

        "SR policy administrative state, true for
        enabled, false for disabled";
    }
}

grouping policy-properties_state {
    description
        "SR policy property grouping";
    leaf oper-state {
        type sr-policy-types:policy-oper-state;
        config false;
        description
            "SR policy operational state";
    }
    leaf transition-count {
        type uint32;
        config false;
        description "Indicates number of up/down transitions";
    }
    leaf up-time {
        type yang:date-and-time;
        config false;
        description "Policy up time in seconds";
    }
    leaf down-time {
        type yang:date-and-time;
        config false;
        description "Policy down time in seconds";
    }
}

grouping policy-properties {
    description
        "SR policy properties";
    uses policy-properties_state;
    uses binding-sid-properties;
    uses forwarding-counters;
}

grouping candidate-path-type {
    description "Candidate path type grouping";
    choice type {
        description
            "Type of candidate paths";
        case explicit {
            description "Candidate path with explicitly defined set/s of segment-li
            uses explicit-path-properties;

```

```
    }
    case dynamic {
      description "Candidate path with dynamic computed segment-lists";
      uses dynamic-path-properties;
    }
  }
}

grouping candidate-paths {
  description "SR policy candidate path grouping";
  container candidate-paths {
    description "SR policy candidate path(s) ";

    list candidate-path {
      key "protocol-origin originator discriminator";
      unique "preference";

      description "SR policy Candidate path(s) list entry";

      leaf protocol-origin {
        type sr-policy-types:protocol-origin-type;
        description
          "Instantiation mechanism used to create the candidate path";
      }
      leaf originator {
        type string;
        description
          "Identifier (concatenation of ASN and node-address) of the node
          that signalled/instantiated the candidate path on headend";
      }
      leaf discriminator {
        type uint32;
        description "Candidate path distinguisher";
      }

      leaf preference {
        type uint32 {
          range "1..65535";
        }
        mandatory true;
        description "Candidate path preference";
      }
      leaf name {
        type string;
        description "Candidate path name";
      }
    }
  }
}
```

```
leaf description {  
    type string;
```

```
        description "Candidate path description";  
    }  
    container binding-sid {  
        if-feature capability-candidate-path-binding-sid;  
        description  
            "Binding segment ID";  
        uses binding-sid_config;  
    }  
  
    uses candidate-path-type;  
    uses candidate-path_state;  
    uses cpath-cmn-properties;  
} }  
}  
  
grouping policies {  
    description "SR policy grouping";  
    container policies {  
        description "SR Policy container";  
  
        list policy {  
            key "color endpoint";  
            unique "name";  
  
            description "SR Policy properties";  
            leaf color {  
                type uint32 {  
                    range "1..4294967295";  
                }  
                description "Color associated with the policy";  
            }  
            leaf endpoint {  
                type inet:ip-address;  
                description "Policy end point IP address";  
            }  
            leaf name {  
                type string {  
                    length "1..59";
```

```

    }
    description "SR policy name";
}
leaf description {
    type string;
    description "Description of the policy";
}
leaf admin-state {
    type sr-policy-types:policy-admin-state;

```

```

    default 'UP';
    description
        "SR policy administrative state, true for
        enabled, false for disabled";
}
leaf priority {
    type uint8;
    default 128;
    description "Priority considered when policy is recomputed due to top
}

uses policy-properties;

uses candidate-paths;
}
}
}

grouping explicit-binding-sid-rules {
    description
        "Grouping for explicit binding sid rules";

    list explicit-binding-sid-rules {
        key "index";
        description
            "Explicit binding sid rules applicable for all policies";
        leaf index {
            type uint32;
            description "Explicit binding SID rules list index";
        }
        leaf rule {
            type sr-policy-types:explicit-binding-sid-rule-type;

```

```

        description "Explicit binding sid rule";
    }
}

augment "/rt:routing" {
    description
        "This augments routing-instance configuration with segment-routing sr-po
    container segment-routing {
        description "Main segment routing container";
        container traffic-engineering {
            description "Traffic-engineering container";

            uses attributes;

            uses policies;

```

```

    }
}

/* Notifications */

notification sr-policy-oper-state-change-event {
    description
        "Notification event when the operational state of the SR policy changes";

    leaf policy-name-ref {
        type leafref {
            path "/rt:routing/sr-policy:segment-routing/sr-policy:traffic-engineerin
        }
        description "Reference to sr-policy name";
    }

    leaf policy-color-ref {
        type leafref {
            path "/rt:routing/sr-policy:segment-routing/sr-policy:traffic-engineerin
        }
        description "Reference to sr-policy color";
    }

    leaf policy-endpoint-ref {

```

```

    type leafref {
      path "/rt:routing/sr-policy:segment-routing/sr-policy:traffic-engineerin
    }
    description "Reference to sr-policy endpoint";
  }

  leaf policy-new-oper-state {
    type sr-policy-types:policy-oper-state;
    description "New operational state of the SR policy";
  }

  leaf policy-down-reason {
    type identityref {
      base sr-policy-types:policy-down-reason;
    }
    description "Down reason if the SR policy's new operational state is down
  }
}

notification sr-policy-candidate-path-change-event {
  description
    "Notification event when candidate path changes for SR policy";
}

```

```

  leaf policy-name-ref {
    type leafref {
      path "/rt:routing/sr-policy:segment-routing/sr-policy:traffic-engineerin
    }
    description "Reference to sr-policy name";
  }

  leaf policy-color-ref {
    type leafref {
      path "/rt:routing/sr-policy:segment-routing/sr-policy:traffic-engineerin
    }
    description "Reference to sr-policy color";
  }

  leaf policy-endpoint-ref {
    type leafref {
      path "/rt:routing/sr-policy:segment-routing/sr-policy:traffic-engineerin
    }
  }

```

```

    description "Reference to sr-policy endpoint";
}

leaf existing-preference {
    type uint32;
    description "Existing candidate path preference";
}

leaf new-preference {
    type uint32;
    description "New candidate path preference";
}
}

notification sr-policy-binding-sid-unavailable-event {
    description
        "Notification event when the binding sid of sr-policy is unavailable";

    leaf policy-name-ref {
        type leafref {
            path "/rt:routing/sr-policy:segment-routing/sr-policy:traffic-engineerin
        }
        description "Reference to sr-policy name";
    }

    leaf policy-color-ref {
        type leafref {
            path "/rt:routing/sr-policy:segment-routing/sr-policy:traffic-engineerin
        }
        description "Reference to sr-policy color";
    }
}

```

```

}

leaf policy-endpoint-ref {
    type leafref {
        path "/rt:routing/sr-policy:segment-routing/sr-policy:traffic-engineerin
    }
    description "Reference to sr-policy endpoint";
}

leaf policy-binding-sid-value-ref {
    if-feature capability-candidate-path-binding-sid;
}

```

```

    type leafref {
      path "/rt:routing/sr-policy:segment-routing/sr-policy:traffic-engineerin
    }
    description "Reference to sr-policy binding-sid value";
  }

  leaf reason {
    type identityref {
      base sr-policy-types:binding-sid-unavailable-reason;
    }
    description
      "Reason why the binding sid is unavailable";
  }
}

notification sr-policy-candidate-path-binding-sid-mismatch-event {
  description
    "Notification event when binding sid of requested candidate path
    is different from the binding sid of the existing candidate path";

  leaf policy-color-ref {
    type leafref {
      path "/rt:routing/sr-policy:segment-routing/sr-policy:traffic-engineerin
    }
    description "Reference to sr-policy color";
  }

  leaf policy-endpoint-ref {
    type leafref {
      path "/rt:routing/sr-policy:segment-routing/sr-policy:traffic-engineerin
    }
    description "Reference to sr-policy endpoint";
  }

  leaf existing-candidate-path-protocol-origin-ref {
    type leafref {
      path "/rt:routing/sr-policy:segment-routing/sr-policy:traffic-engineerin

```

```

  }
  description "Reference to existing candidate path protocol origin";
}

```



```

leaf existing-candidate-path-preference-ref {
  type leafref {
    path "/rt:routing/sr-policy:segment-routing/sr-policy:traffic-engineering
  }
  description "Reference to existing candidate path preference";
}

leaf existing-candidate-path-binding-sid-dataplane-ref {
  if-feature capability-candidate-path-binding-sid;
  type leafref {
    path "/rt:routing/sr-policy:segment-routing/sr-policy:traffic-engineering
  }
  description "Reference to existing candidate path binding sid dataplane ty
}

leaf existing-candidate-path-binding-sid-value-ref {
  if-feature capability-candidate-path-binding-sid;
  type leafref {
    path "/rt:routing/sr-policy:segment-routing/sr-policy:traffic-engineering
  }
  description "Reference to existing candidate path binding sid value";
}

leaf conflicting-candidate-path-protocol-origin {
  type uint8;
  description "Conflicting candidate path protocol origin";
}

leaf conflicting-candidate-path-preference {
  type uint32;
  description "Conflicting candidate path preference";
}

leaf conflicting-candidate-path-binding-sid-dataplane {
  type sr-policy-types:dataplane-type;
  description "Conflicting candidate path binding sid dataplane type";
}

leaf conflicting-candidate-path-binding-sid-value {
  type sr-policy-types:sid-value-type;
  description "Conflicting candidate path binding sid value";
}
}

```

```

/* Features */

feature capability-candidate-path-binding-sid {
  description
    "This feature enables the capability of specifying binding-sid
    for a candidate path.";
}
} // module

<CODE ENDS>

```

Figure 6: ietf-sr-policy.yang

7. Security Considerations

The configuration, state, and notification data defined using YANG data models in this document are likely to be accessed via the protocols such as NETCONF [[RFC6241](#)] etc.

Hence, YANG implementations MUST comply with the security requirements specified in [section 15 of \[RFC6020\]](#). Additionally, NETCONF implementations MUST comply with the security requirements specified in [sections 2.2, 2.3 and 9 of \[RFC6241\]](#) as well as [section 3.7 of \[RFC8341\]](#).

8. IANA Considerations

This document requests the registration of the following URIs in the IETF "XML registry" [[RFC3688](#)]:

URI	Registrant	XML
urn:ietf:params:xml:ns:yang:ietf-sr-policy-types	The IESG	N/A
urn:ietf:params:xml:ns:yang:ietf-sr-policy	The IESG	N/A

This document requests the registration of the following YANG modules in the "YANG Module Names" registry [[RFC6020](#)]:

Name	Namespace	Prefix	Reference
ietf-sr-policy-types	urn:ietf:params:xml:ns:yang:ietf-sr-policy-types	sr-policy-types	This document
ietf-sr-policy	urn:ietf:params:xml:ns:yang:ietf-sr-policy	sr-policy	This document

9. Acknowledgments

The authors of this document/YANG model would like to acknowledge the contributions/reviews by Johnson Thomas, Clarence Filsfils, Siva Sivabalan, Tarek Saad, Kris Michielsen, Dhanendra Jain, Ketan Talaulikar, Bhupendra Yadav, and Bruno Decraene.

10. References

10.1. Normative References

- [I-D.ietf-spring-segment-routing-policy]
Filsfils, C., Talaulikar, K., Voyer, D., Bogdanov, A., and P. Mattes, "Segment Routing Policy Architecture", [draft-ietf-spring-segment-routing-policy-09](#) (work in progress), November 2020.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", [BCP 14](#), [RFC 2119](#), DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC3688] Mealling, M., "The IETF XML Registry", [BCP 81](#), [RFC 3688](#), DOI 10.17487/RFC3688, January 2004, <<https://www.rfc-editor.org/info/rfc3688>>.
- [RFC6020] Bjorklund, M., Ed., "YANG - A Data Modeling Language for

the Network Configuration Protocol (NETCONF)", [RFC 6020](#), DOI 10.17487/RFC6020, October 2010, <<https://www.rfc-editor.org/info/rfc6020>>.

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

Raza, et al.

Expires October 9, 2021

[Page 45]

Internet-Draft

YANG Data Model for SR Policy

April 2021

- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in [RFC 2119](#) Key Words", [BCP 14](#), [RFC 8174](#), DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.
- [RFC8341] Bierman, A. and M. Bjorklund, "Network Configuration Access Control Model", STD 91, [RFC 8341](#), DOI 10.17487/RFC8341, March 2018, <<https://www.rfc-editor.org/info/rfc8341>>.
- [RFC8342] Bjorklund, M., Schoenwaelder, J., Shafer, P., Watsen, K., and R. Wilton, "Network Management Datastore Architecture (NMDA)", [RFC 8342](#), DOI 10.17487/RFC8342, March 2018, <<https://www.rfc-editor.org/info/rfc8342>>.
- [RFC8402] Filsfils, C., Ed., Previdi, S., Ed., Ginsberg, L., Decraene, B., Litkowski, S., and R. Shakir, "Segment Routing Architecture", [RFC 8402](#), DOI 10.17487/RFC8402, July 2018, <<https://www.rfc-editor.org/info/rfc8402>>.
- [RFC8407] Bierman, A., "Guidelines for Authors and Reviewers of Documents Containing YANG Data Models", [BCP 216](#), [RFC 8407](#), DOI 10.17487/RFC8407, October 2018, <<https://www.rfc-editor.org/info/rfc8407>>.

[10.2.](#) Informative References

- [I-D.ietf-idr-segment-routing-te-policy]
Previdi, S., Filsfils, C., Talaulikar, K., Mattes, P., Rosen, E., Jain, D., and S. Lin, "Advertising Segment Routing Policies in BGP", [draft-ietf-idr-segment-routing-te-policy-11](#) (work in progress), November 2020.

Authors' Addresses

Kamran Raza (editor)
Cisco Systems
Email: skraza@cisco.com

Robert Sawaya
Cisco Systems
Email: rsawaya@cisco.com

Raza, et al.

Expires October 9, 2021

[Page 46]

Internet-Draft

YANG Data Model for SR Policy

April 2021

Zhuang Shunwan
Huawei Technologies
Email: zhuangshunwa@huawei.com

Daniel Voyer
Bell Canada
Email: daniel.voyer@bell.ca

Muhammad Durrani
Equinix
Email: mdurrani@equinix.com

Satoru Matsushima
SoftBank
Email: satoru.matsushima@g.softbank.co.jp

Vishnu Pavan Beeram
Juniper Networks
Email: vbeeram@juniper.net

