

INTERNET-DRAFT
Intended status: Proposed Standard

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

Expires: May 7, 2019

November 8, 2018

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

Abstract

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

Status of this Memo

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

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

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

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

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

INTERNET-DRAFT

YANG Model for Tunnel Policy

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

INTERNET-DRAFT

YANG Model for Tunnel Policy

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 tunnelPolycys
|   +--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.

INTERNET-DRAFT

YANG Model for Tunnel Policy

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

```

INTERNET-DRAFT

YANG Model for Tunnel Policy

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 tnImbaseTnlPolicyType {  
  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 tnImbaseSelTnlType {
type enumeration {
enum "invaild" {
description
"Search for invalid tunnels.";
}
enum "lsp" {
description
"Search for LDP LSPs.";
}
enum "cr-lsp" {
description
"Search for CR-LSPs.";
}
enum "gre" {
description
"Search for GREs.";
}
enum "ldp" {
description
"Search for LDP LSPs.";
}
enum "bgp" {

```

```

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

```

```

}

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

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

```

```

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

```

```

}
description
  "tunnel selector type";
}

```

```

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

```

Tunnel policies can be classified into two modes:

Select-seq: The system selects a tunnel for an application program based on the tunnel type priorities defined in the tunnel policy.

Tunnel binding: The system selects only a specified tunnel for an application program.

The two modes are mutually exclusive.

Configuration example:

```

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

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

```

```

default "true";
description

```

```

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

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

list tunnelPolicy {
    key "tnlPolicyName";

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

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

```

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

  leaf tnlPolicyType {
    type tnlmbaseTnlPolicyType;
    default "invalid";
    description
      "Tunnel policy type. The available options are sel-seq,
      binding, and invalid. A tunnel policy can be configured
      with only one policy type.";
  }

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



```

type boolean;
default "false";
description
    "Do not check whether the destination address of the

```

```

        TE tunnel matches the destination address specified
        in the tunnel policy.";
    }
    leaf isIncludeLdp {
        type boolean;
        must "(../isIncludeLdp='true' and not "
            + "(../downSwitch='true')) or "
            + "../isIncludeLdp='false'";
        default "false";
        description
            "Is loadbalance with LDP";
    }
    container tpTunnels {
        description
            "List of tunnels available for an application.";
        list tpTunnel {
            key "tunnelName";
            min-elements "1";
            max-elements "16";
            description
                "Tunnel.";
            leaf tunnelName {
                type string {
                    length "1..47";
                }
                description
                    "Name of the specified tunnel.";
            }
        }
    }
}

container tnlSelSeqs {
    when "not(../tnlPolicyType='invalid' or "
        + "../tnlPolicyType='tnlBinding')";
    must "not(../tnlPolicyType='tnlSelectSeq') or "
        + "(../tnlPolicyType='tnlSelectSeq' and "
        + "count(tnlSelSeq)>=1)";
}

```

```

description
  "Sequence in which different types of tunnels are
  selected.
  If the value is INVALID, no tunnel type has been
  configured.";
container tnlSelSeq {
  when "not(..../tnlPolicyType='invalid' or "
    + "..../tnlPolicyType='tnlBinding') or "
    + "..../tnlPolicyType='tnlSelectSeq'";
  presence "create tnlSelSeq";
  description
    "Sequence in which different types of tunnels are

```

```

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

```

```

        has been configured.";
    }
    leaf selTnlType3 {
        when "not(../selTnlType1='invalid' or "
            + "../selTnlType2='invalid')";
        type tnImbaseSelTnlType;
        default "invalid";
        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='invalid' or "
            + "../selTnlType2='invalid' or "
            + "../selTnlType3='invalid')";
        type tnImbaseSelTnlType;
        default "invalid";
        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='invalid' or "
            + "../selTnlType2='invalid' or "
            + "../selTnlType3='invalid' or "
            + "../selTnlType4='invalid')";
        type tnImbaseSelTnlType;
        default "invalid";
        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='invalid' or "
            + "../selTnlType2='invalid' or "
            + "../selTnlType3='invalid' or "
            + "../selTnlType4='invalid' or "
            + "../selTnlType5='invalid')";
        type tnImbaseSelTnlType;

```

```

        default "invalid";
        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]))|'
```

INTERNET-DRAFT

YANG Model for Tunnel Policy

```
        + '([^-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

- [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.

INTERNET-DRAFT

YANG Model for Tunnel Policy

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

INTERNET-DRAFT

YANG Model for Tunnel Policy

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.

