OPSAWG
Internet-Draft
Intended status: Standards Track
Expires: April 19, 2021

S. Barguil
O. Gonzalez de Dios, Ed.
Telefonica
M. Boucadair, Ed.
Orange
L. Munoz
Vodafone
A. Aguado
Nokia

October 16, 2020

A Layer 3 VPN Network YANG Model
draft-ietf-opsawg-l3sm-l3nm-05

Abstract

This document defines a L3VPN Network YANG Model (L3NM) that can be used to manage the provisioning of Layer 3 Virtual Private Network (VPN) services within a Service Provider's network. The model provides a network-centric view of L3VPN services.

L3NM is meant to be used by a Network Controller to derive the configuration information that will be sent to relevant network devices. The model can also facilitate the communication between a service orchestrator and a network controller/orchestrator.

Editorial Note (To be removed by RFC Editor)

Please update these statements within the document with the RFC number to be assigned to this document:

- "This version of this YANG module is part of RFC XXXX;"
- "RFC XXXX: Layer 3 VPN Network Model";
- reference: RFC XXXX

Also, please update the "revision" date of the YANG module.

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 April 19, 2021.

Table of Contents

1. Introduction

[RFC8299] defines a L3VPN Service YANG data Model (L3SM) that can be used for communication between customers and network operators. Such model is focused on describing the customer view of the Virtual Private Network (VPN) services, and provides an abstracted view of the customer's requested services. That approach limits the usage of the L3SM module to the role of a Customer Service Model, according to the terminology defined in [RFC8309].

This document defined a YANG module called L3VPN Network Model (L3NM). The L3NM is aimed at providing a network-centric view of Layer 3 (L3) VPN Services. This data model can be used to facilitate communication between the service orchestrator (or a network operator) and the network controller/orchestrator by allowing for more network-centric information to be included. It enables further capabilities, such as resource management or to serve as a multi-domain orchestration interface, where logical resources (such as route targets or route distinguishers) must be synchronized.

This document uses the common VPN YANG module defined in [I-D.ietf-opsawg-vpn-common].

This document does not obsolete, but uses, the definitions in [RFC8299]. These two modules are used for similar objectives but with different scopes and views.

The L3NM YANG module is initially built with a prune and extend approach, taking as a starting points the YANG module described in [RFC8299]. Nevertheless, this module is not defined as an augment to L3SM because a specific structure is required to meet network-oriented L3 needs.

Some of the information captured in the L3SM can be passed by the Orchestrator in the L3NM (e.g., customer) or be used to fed some of the L3NM attributes (e.g., actual forwarding policies). Some of the information captured in L3SM may be maintained locally within the Orchestrator; which is in charge of maintaining the correspondence between a Customer view and its network instantiation. Likewise, some of the information captured and exposed using L3NM can fed the service layer (e.g., capabilities) to derive L3SM and drive VPN service order handling.

The L3NM does not attempt to address all deployment cases especially those where the L3VPN connectivity is supported through the coordination of different VPNs in different underlying networks. More complex deployment scenarios involving the coordination of different VPN instances and different technologies to provide end-to-end VPN connectivity are addressed by a complementary YANG model defined in [I-D.evenwu-opsawg-yang-composed-vpn].

L3NM focuses on BGP PE-based Layer 3 VPNs as described in [RFC4026][RFC4110][RFC4364] and Multicast VPNs as described in [RFC6037][RFC6513][RFC7988].

The YANG data model in this document conforms to the Network Management Datastore Architecture (NMDA) defined in [RFC8342].

2. Requirements Language

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

3. Terminology

This document assumes that the reader is familiar with the contents of [RFC6241], [RFC7950], [RFC8299], [RFC8309], and [RFC8453] and uses the terminology defined in those documents.

The meaning of the symbols in tree diagrams is defined in [RFC8340].

The document is aimed at modeling BGP PE-based VPNs in a service provider network, so the terms defined in [RFC4026] and [RFC4176] are used.

This document makes use of the following terms:

- Layer 3 VPN Customer Service Model (L3SM): A YANG module that describes the requirements of a L3VPN that interconnects a set of sites from the point of view of the customer. The customer service model does not provide details on the service provider network. The L3VPN Customer Service model is defined in [RFC8299].

- Layer 3 VPN Service Network Model (L3NM): A YANG module that describes a VPN Service in the service provider network. It contains information of the Service Provider network and might include allocated resources. It can be used by network controllers to manage and control the VPN Service configuration in the Service Provider network. The YANG module can be consumed by a Service Orchestrator to request a VPN Service to a Network controller.

- Service Orchestrator: A functional entity that interacts with the customer of a L3VPN. The Service Orchestrator interacts with the customer using L3SM. The Service Orchestrator is responsible of the Customer Edge (CE) - the Provider Edge (PE) attachment circuits, the PE selection, and requesting the VPN service to the network controller.

- Network Orchestrator: A functional entity that is hierarchically intermediate between Service Orchestrator and Network Controllers. A network orchestrator can manage one or several Network Controllers.

- Network Controller: A functional entity responsible for the control and management of the service provider network.

- VPN node: An abstraction that represents a set of policies applied on a PE and that belong to a single VPN service. A VPN service involves one or more VPN nodes. As it is an abstraction, the network controller will take on how to implement a VPN node. For example, typically, in a BGP-based VPN, a VPN node could be mapped into a Virtual Routing and Forwarding (VRF).

- VPN network access: An abstraction that represents the network interfaces that are associated to a given VPN node. Traffic coming from the VPN network access belongs to the VPN. The attachment circuits (bearers) between CEs and PEs are terminated

in the VPN network access. A reference to the bearer is maintained to allow keeping the link between L3SM and L3NM.

- VPN Site: A VPN customer's location that is connected to the Service Provider network via a CE-PE link, which can access at least one VPN [RFC4176].

- VPN Service Provider (SP): A Service Provider that offers VPN-related services [RFC4176].

- Service Provider (SP) Network: A network that is able to provide VPN-related services.

4. L3NM Reference Architecture

Figure 1 depicts the reference architecture for L3NM. The figure is an expansion of the architecture presented in Section 5 of [RFC8299] and decomposes the box marked "orchestration" in that figure into three separate functional components called "Service Orchestration", "Network Orchestration", and "Domain Orchestration".

Although some deployments may choose to construct a monolithic orchestration component (covering both service and network matters), this document advocates for a clear separation between service and network orchestration components for the sake of better flexibility. Such design adheres to the L3VPN reference architecture defined in Section 1.3 of [RFC4176]. The above separation relies upon a dedicated communication interface between these components and appropriate YANG module that reflect network-related information (that is hidden to customers).

The intelligence for translating customer-facing information into network-centric one is implementation specific.

The terminology from [RFC8309] is introduced to show the distinction between the "Customer Service Model", the "Service Delivery Model", the "Network Configuration Model", and the "Device Configuration Model". In that context, the "Domain Orchestration" and "Config Manager" roles may be performed by "Controllers".

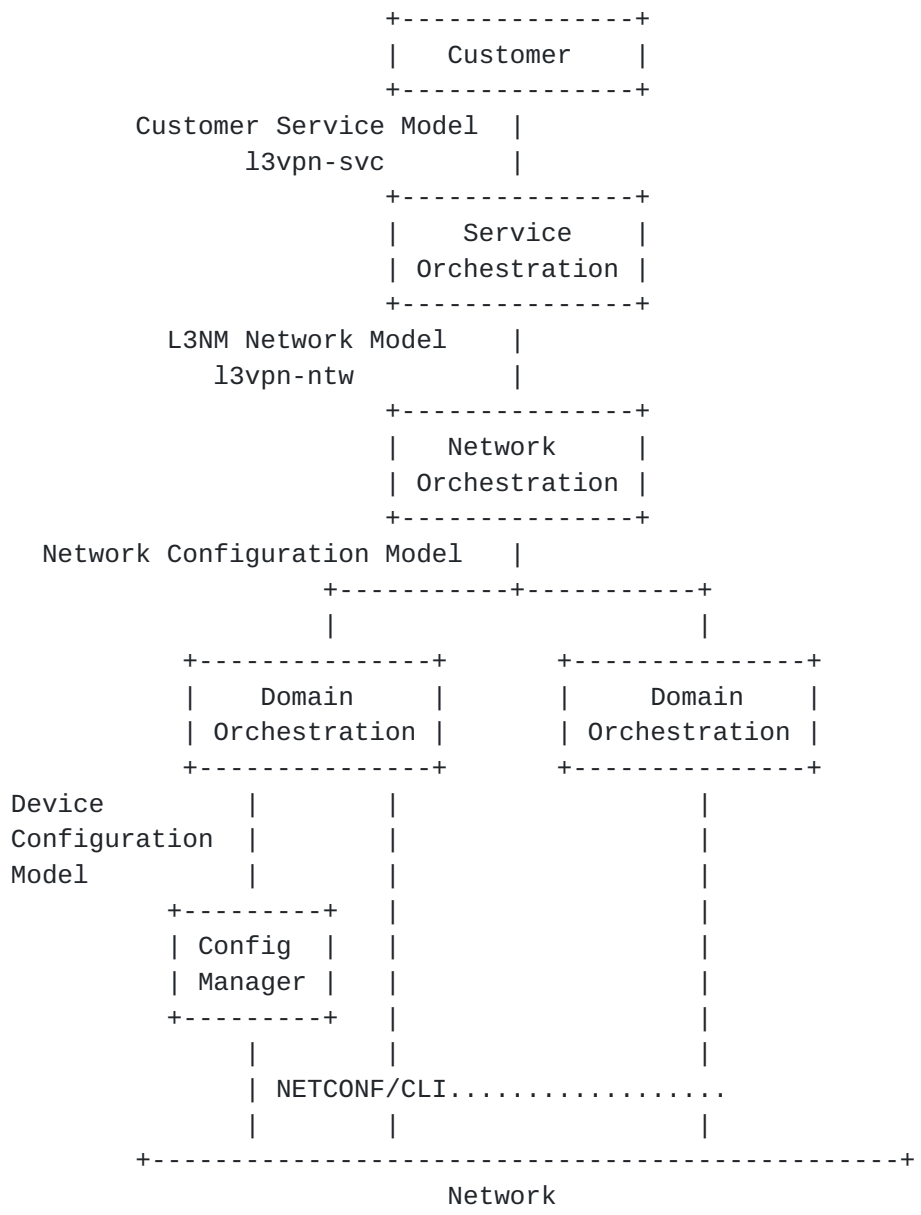


Figure 1: Reference Architecture

The L3SM and the L3NM may also be used in the context of the ACTN architecture [RFC8453]. Figure 2 shows the Customer Network Controller (CNC), the Multi-Domain Service Coordinator (MDSC), and the Provisioning Network Controller (PNC). It also shows the interfaces between these functional blocks: the CNC-MDSC Interface (CMI), the MDSC-PNC Interface (MPI), and the Southbound Interface (SBI).

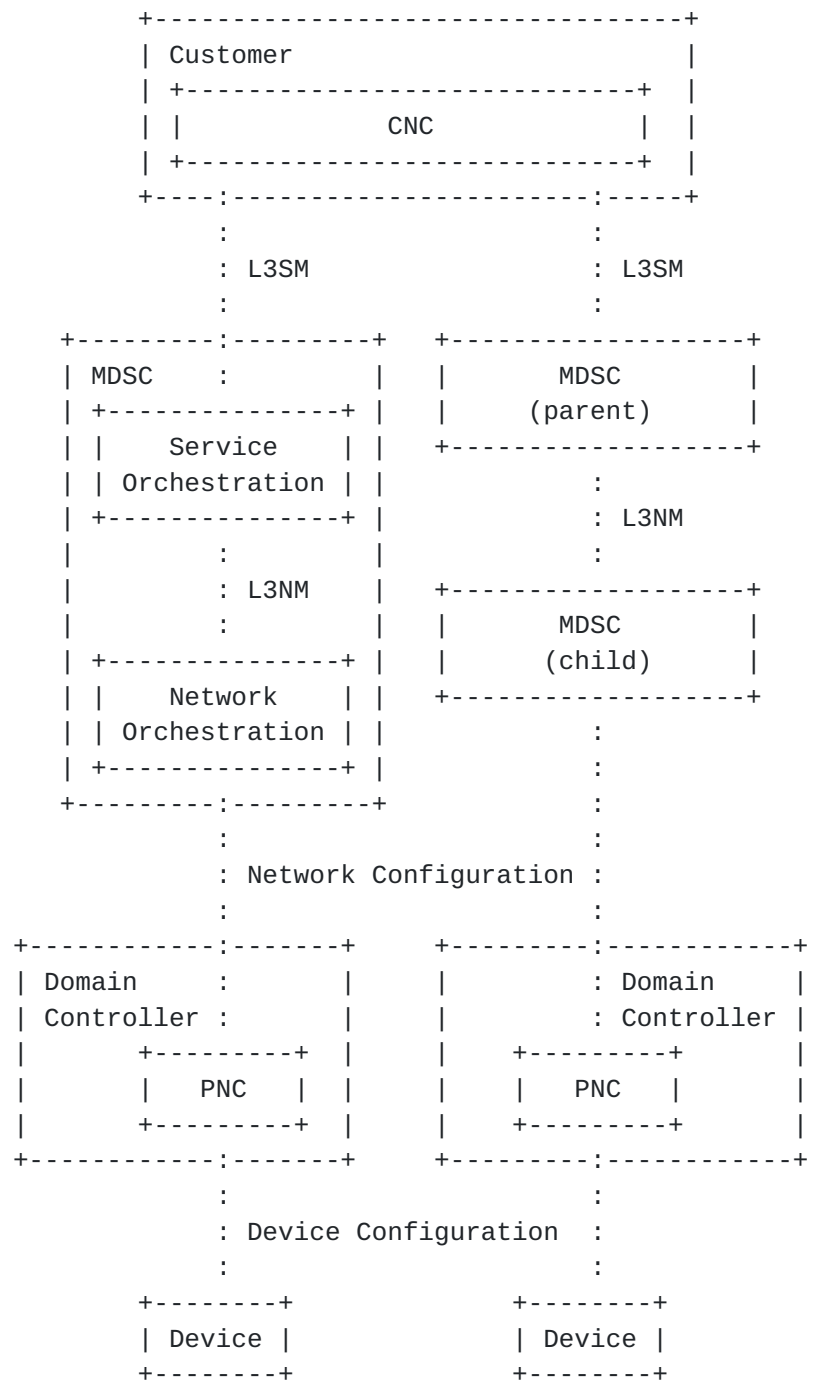


Figure 2: L3SM and L3NM in the Context of ACTN

5. Relation with other YANG Models

The "ietf-vpn-common" module [I-D.ietf-opsawg-vpn-common] includes a set of identities, types, and groupings that are meant to be reused by VPN-related YANG modules independently of the layer (e.g., Layer 2, Layer 3) and the type of the module (e.g., network model, service

model) including future revisions of existing models (e.g., [RFC8299] or [RFC8466]). The L3NM reuses these common types and grouping.

In order to avoid data duplication and to ease passing data between layers when required (service layer to network layer and vice versa), early versions of the L3NM reused many of the data nodes that are defined in [RFC8299]. Nevertheless, that approach was abandoned in favor of the "ietf-vpn-common" module because that design was interpreted as if the deployment of L3NM depends on L3SM, while this is not the case. For example, a Service Provider may decide to use the L3NM to build its L3VPN services without exposing the L3SM.

As discussed in Section 4, the L3NM YANG module is meant to manage L3VPN services within a Service Provider network. The module provides a network view of the service. Such view is only visible within the Service Provider and is not exposed outside (to customers, for example). The following discusses how L3NM interfaces with other YANG modules:

L3SM: L3NM is not a Customer Service Model.

The internal view of the service (L3NM) may be mapped to an external view which is visible to Customers : L3VPN Service YANG data Model (L3SM) [RFC8299].

Typically, the L3NM can be fed with inputs that are requested by Customers, typically, relying upon a L3SM template. Concretely, some parts of the L3SM module can be directly mapped into L3NM while other parts are generated as a function of the requested service and local guidelines. Some other parts are local to the Service Provider and do not map directly to L3SM.

Note that the use of L3NM within a Service Provider does not assume nor preclude exposing the VPN service via L3SM. This is deployment-specific. Nevertheless, the design of L3NM tries to align as much as possible with the features supported by the L3SM to ease grafting both L3NM and L3SM for the sake of highly automated VPN service provisioning and delivery.

Network Topology Modules: A L3VPN involves nodes that are part of a topology managed by the Service Provider Backbone network. Such topology can be represented as using the network topology module in [RFC8345].

Device Modules: L3NM is not a device model.

Once a global VPN service is captured by means of L3NM, the actual activation and provisioning of the VPN service will involve a

variety of device modules to tweak the required functions for the delivery of the service. These functions are supported by the VPN nodes and can be managed using device YANG modules. A non-comprehensive list of such device YANG modules is provided below:

* Routing management [RFC8349].

* BGP [I-D.ietf-idr-bgp-model].

* PIM [I-D.ietf-pim-yang].

* NAT management [RFC8512].

* QoS management [I-D.ietf-rtgwg-qos-model].

* ACLs [RFC8519].

How L3NM is used to derive device-specific actions is implementation-specific.

6. Sample Uses of the L3NM Data Model

6.1. Enterprise Layer 3 VPN Services

Enterprise L3VPNs are one of the most demanded services for carriers, and therefore, L3NM can be useful to automate the tasks of provisioning and maintenance of these VPNs. Templates and batch processes can be built, and as a result many parameters are needed for the creation from scratch of a VPN that can be abstracted to the upper SDN layer and little manual intervention will be still required.

Also common addition/removal of sites of an existing customer VPN can benefit of using L3NM, by creation of workflows that either prune or add nodes as required from the network data model object.

6.2. Multi-Domain Resource Management

The implementation of L3VPN services which span across administratively separated domains (i.e., that are under the administration of different management systems or controllers) requires some network resources to be synchronized between systems. Particularly, there are two resources that must be orchestrated and manage to avoid asymmetric (non-functional) configuration, or the usage of unavailable resources.

For example, RTs shall be synchronized between PEs. When every PE is controlled by the same management system, RT allocation can be

performed by the system. In cases where the service spans across multiple management systems, this task of allocating RTs has to be aligned across the domains, therefore, the service model must provide a way to specify RTs. In addition, RDs must also be synchronized to avoid collisions in RD allocation between separate systems. An incorrect allocation might lead to the same RD and IP prefixes being exported by different PE routers.

6.3. Management of Multicast Services

Multicast services over L3VPN can be implemented either using dual PIM MVPNs (also known as, Draft Rosen model) [RFC4364] or multiprotocol BGP (MBGP)-based MVPNs[RFC6513][RFC6514]. Both methods are supported and equally effective, but the main difference is that MBGP-based MVPN does not require multicast configuration on the service provider backbone. MBGP MVPNs employ the intra-autonomous system BGP control plane and PIM sparse mode as the data plane. The PIM state information is maintained between the PE routers using the same architecture that is used for unicast VPNs.

On the other hand, Draft Rosen has limitations such as reduced options for transport, control plane scalability, availability, operational inconsistency, and the need of maintaining state in the backbone. Because of this, MBGP MVPN is the architectural model that has been taken as the base for implementing multicast service on L3VPN. In this scenario, BGP auto discovery is used to discover MVPN PE members and the customer PIM signaling is sent across provider core through MP-BGP. The multicast traffic is transported on MPLS P2MP LSPs. All of the previous information is carried in the MCAST-VPN BGP NRLI.

7. Description of the L3NM YANG Module

The L3NM ('ietf-l3vpn-ntw') is defined to manage L3VPNs in a service provider network. In particular, the 'ietf-l3vpn-ntw' module can be used to create, modify, and retrieve L3VPN Services of a network.

7.1. Overall Structure of the Module

The 'ietf-l3vpn-ntw' module uses two main containers: 'vpn-services' and 'vpn-profiles' (see Figure 3).

The 'vpn-services' container maintains the set of VPN services managed within the service provider's network. 'vpn-service' is the data structure that abstracts a VPN service (Section 7.3).

The 'vpn-profiles' container is used by the provider to maintain a set of common VPN profiles that apply to one or several VPN services (Section 7.2).

```
module: ietf-l3vpn-ntw
  +--rw l3vpn-ntw
     +--rw vpn-profiles
     |  ...
     +--rw vpn-services
         +--rw vpn-service* [vpn-id]
            ...
```

Figure 3: Overall L3NM Tree Structure

7.2. VPN Profiles

The 'vpn-profiles' container (Figure 4) allows the network provider to define and maintain a set of common VPN profiles [I-D.ietf-opsawg-vpn-common] that apply to one or several VPN services. The exact definition of the profiles is local to each network provider.

This document does not make any assumption about the exact definition of these profiles. How such profiles are defined is deployment specific. The model only includes an identifier to these profiles to ease identifying local policies when building a VPN service. As shown in Figure 4, the following identifiers can be included:

- 'cloud-identifier': This identifier refers to a cloud service.
- 'encryption-profile-identifier': An encryption profile refers to a set of policies related to the encryption scheme(s) and setup that can be applied when building and offering a VPN service.
- 'qos-profile-identifier': A QoS profile refers to as set of policies such as classification, marking, and actions (e.g., [RFC3644]).
- 'bfd-profile-identifier': A Bidirectional Forwarding Detection (BFD) profile refers to a set of BFD [RFC5880] policies that can be invoked when building a VPN service.
- 'forwarding-profile-identifier': A forwarding profile refers to the policies that apply to the forwarding of packets conveyed within a VPN. Such policies may consist at applying Access Control Lists (ACLs).

- 'routing-profile-identifier': A routing profile refers to a set of routing policies that will be invoked (e.g., BGP policies).

```
      +--rw l3vpn-ntw
         +--rw vpn-profiles
         |  +--rw valid-provider-identifiers
         |     +--rw cloud-identifier* [id] {cloud-access}?
         |     |  +--rw id    string
         |     +--rw encryption-profile-identifier* [id]
         |     |  +--rw id    string
         |     +--rw qos-profile-identifier* [id]
         |     |  +--rw id    string
         |     +--rw bfd-profile-identifier* [id]
         |     |  +--rw id    string
         |     +--rw forwarding-profile-identifier* [id]
         |     |  +--rw id    string
         |     +--rw routing-profile-identifier* [id]
         |        +--rw id    string
         +--rw vpn-services
            ...
```

Figure 4: VPN Profiles Subtree Structure

7.3. Modeling a Layer 3 VPN Service

The 'vpn-service' is the data structure that abstracts a VPN service in the service provider network. Each 'vpn-service' is uniquely identified by an identifier: 'vpn-id'. Such 'vpn-id' is only meaningful locally within the Network controller.

In order to facilitate the identification of the service, 'customer-name' and 'description' attributes may be provided.

The main 'vpn-service' parameters are:

- 'status': Allows the control of the operative and administrative status of the service as a whole.

- 'vpn-id': Is an identifier that is used to uniquely identify the L3VPN Service within L3NM scope.

- 'l3sm-vpn-id': Refers to an identifier of L3SM service. This identifier allows to easily correlate the (network) service as built in the network with a service order.

- 'vpn-service-topology': Indicates the network topology for the service: Hub-Spoke, Any-to-Any, and Custom. The deployment on the

network is defined by the correct usage of import and export profiles

- 'vpn-type': Indicate the VPN service signaling type.

- 'ie-profiles': Defines reusable import/export policies for the same 'vpn-service'. More details are provided in Section 7.3.2.

- 'underlay-transport': Describes the preference for the transport technology to carry the traffic of the VPN service (Section 7.3.3).

The 'vpn-node' is an abstraction that represents a set of policies applied to a network node and that belong to a single 'vpn-service'. A VPN service is typically built by adding instances of 'vpn-node' to the 'vpn-nodes' container.

A 'vpn-node' contains 'vpn-network-accesses', which are the interfaces attached to the VPN by which the customer traffic is received. Therefore, the customer sites are connected to the 'vpn-network-accesses'.

Note that, as this is a network data model, the information about customers sites is not required in the model. Such information is rather relevant in the L3SM.

```
  +--rw l3vpn-ntw
     +--rw vpn-profiles
     |  ...
     +--rw vpn-services
        +--rw vpn-service* [vpn-id]
           +--rw status
           |  ...
           +--rw vpn-id                 vpn-common:vpn-id
           +--rw vpn-name?              string
           +--rw vpn-description?       string
           +--rw customer-name?         string
           +--rw l3sm-vpn-id?           vpn-common:vpn-id
           +--rw vpn-type?              identityref
           +--rw vpn-service-topology?  identityref
           +--rw ie-profiles
           |  ...
           +--rw underlay-transport
           |  ...
           +--rw vpn-nodes
              ...
```

Figure 5: VPN Services Subtree Structure

7.3.1. Service Status

The L3NM allows to track service status ('status') of a given VPN service (Figure 6). Both operational and administrative status are maintained together with a timestamp. For example, a service can be created but not put into effect.

'admin' and 'ops' status can be used as trigger to detect service anomalies. For example, a service that is declared at the service layer as active but still inactive at the network layer is an indication that network provision actions are needed to align the observed service with the expected service status.

```
  +--rw l3vpn-ntw
     +--rw vpn-profiles
     |  ...
     +--rw vpn-services
        +--rw vpn-service* [vpn-id]
           +--rw status
           |  +--rw admin-status
           |  |  +--rw status?         identityref
           |  |  +--rw last-updated?   yang:date-and-time
           |  +--ro oper-status
           |     +--ro status?         identityref
           |     +--ro last-updated?   yang:date-and-time
           ...
```

Figure 6: VPN Service Status Subtree Structure

7.3.2. Concept of Import/Export Profiles

The import and export profiles construct contains a list with information related with route target and distinguishers (RTs and RDs), grouped and identified by 'ie-profile-id'. The identifier is then referenced in one or multiple 'vpn-nodes' so the controller can identify RTs and RDs to be configured for a given VRF. The subtree is shown in Figure 7.

```
+--rw l3vpn-ntw
   +--rw vpn-profiles
   |  ...
   +--rw vpn-services
      +--rw vpn-service* [vpn-id]
         +--rw vpn-id                 vpn-common:vpn-id
         +  ...
         +--rw ie-profiles
         |  +--rw ie-profile* [ie-profile-id]
         |     +--rw ie-profile-id    string
         |     +--rw rd?              union
         |     +--rw vpn-targets
         |        +--rw vpn-target* [id]
         |        |  +--rw id                   int8
         |        |  +--rw route-targets* [route-target]
         |        |  |  +--rw route-target    rt-types:route-target
         |        |  +--rw route-target-type
         |        |          rt-types:route-target-type
         |        +--rw vpn-policies
         |           +--rw import-policy?   string
         |           +--rw export-policy?   string
         +--rw vpn-nodes
            +--rw vpn-node* [ne-id]
               +--rw ne-id                      string
               ...
               +--rw vpn-targets
               |  +--rw vpn-target* [id]
               |  |  +--rw id                   int8
               |  |  +--rw route-targets* [route-target]
               |  |  |  +--rw route-target    rt-types:route-target
               |  |  +--rw route-target-type
               |  |          rt-types:route-target-type
               |  +--rw vpn-policies
               |     +--rw import-policy?   string
               |     +--rw export-policy?   string
               ...
```

Figure 7: Subtree Structure of Import/Export Profiles

7.3.3. Underlay Transport

The model allows to indicate a preference for the underlay transport technology when activating a L3VPN service (Figure 8). This preference is especially useful in networks with multiple domains and NNI types. This version of the YANG module supports these options: BGP, LDP, GRE, SR, SR-TE, and RSVP-TE as underlay transport mechanisms. Other profiles can be defined in the future.

```
  +--rw l3vpn-ntw
     +--rw vpn-profiles
     |  ...
     +--rw vpn-services
        +--rw vpn-service* [vpn-id]
           +--rw vpn-id                  vpn-common:vpn-id
           +  ...
           +--rw underlay-transport
           |  +--rw type*   identityref
           +--rw vpn-nodes
              +--rw vpn-node* [ne-id]
              ...
```

Figure 8: Subtree Structure of the Underlying Transport

7.3.4. VPN Node

The 'vpn-node' is an abstraction that represents a set of common policies applied on a given network node (tipcally, a PE) and belong to one L3VPN service. In order to indicate the network nodes where the 'vpn-node' applies, the 'ne-id' must be indicated. The 'vpn-node' includes a parameter to indicate the network node on which it is applied. In the case that the 'ne-id' points to a specific PE, the 'vpn-node' will likely be mapped into a VRF in the node. However, the model also allows to point to an abstract node. In this case, the network controller will decide how to split the 'vpn-node' into VRFs. Some 'vpn-node' parameters are listed below:

- local-autonomous-system: Refers to the autonomous system number that is locally configured in the instance. It can be overwritten for specific purposes in the CE-PE BGP session.

- maximum-routes: Set the maximum number of prefixes allowed in the 'vpn-node' instance. This value is typically set in the service request.

- 'rd' and 'vpn-targets': For the cases the logical resources are managed outside the network controller, the model allows to explicitely indicate the logical resources such as Route targets (RTs) and Route Distinguishers (RDs) (RT,RD).

- Multicast: Enable multicast traffic inside the VPN. Refer to Section 7.3.4.3.

Under the VPN Node ('vpn-node') container, VPN Network Accesses ('vpn-network-access') can be created. The VPN Network Access represents the point to which sites are connected. Note that, unlike in L3SM, the L3NM does not need to model the customer site, only the

points where the traffic from the site are received (i.e., the PE side of PE-CE connections). Hence, the VPN Network access contains the connectivity information between the provider's network and the customer premises. The VPN profiles ('vpn-profiles') have a set of routing policies than can be applied during the service creation.

The L3NM allows to track the status ('status') of the nodes involved in a VPN service. Both operational and administrative status are maintained. Mismatch between an administrative status vs. the operational status can be used as trigger to detect anomalies.

```
  +--rw l3vpn-ntw
     +--rw vpn-profiles
     |  ...
     +--rw vpn-services
        +--rw vpn-service* [vpn-id]
           +--rw vpn-id                  vpn-common:vpn-id
           ...
           +--rw vpn-nodes
              +--rw vpn-node* [ne-id]
                 +--rw vpn-node-id?               union
                 +--rw local-autonomous-system?   inet:as-number
                 +--rw description?               string
                 +--rw ne-id                      string
                 +--rw router-id?                 inet:ip-address
                 +--rw address-family?
                 |       vpn-common:address-family
                 +--rw node-role?                 identityref
                 +--rw rd?                        union
                 +--rw vpn-targets
                 |  +--rw vpn-target* [id]
                 |  |  +--rw id                   int8
                 |  |  +--rw route-targets* [route-target]
                 |  |  |  +--rw route-target    rt-types:route-target
                 |  |  +--rw route-target-type
                 |  |          rt-types:route-target-type
                 |  +--rw vpn-policies
                 |     +--rw import-policy?   string
                 |     +--rw export-policy?   string
                 +--rw status
                 |  +--rw admin-status
                 |  |  +--rw status?         identityref
                 |  |  +--rw last-updated?   yang:date-and-time
                 |  +--ro oper-status
                 |     +--ro status?         identityref
                 |     +--ro last-updated?   yang:date-and-time
                 +--rw node-ie-profile?           leafref
                 +--rw groups
```

```
|  +--rw group* [group-id]
|     +--rw group-id    string
+--rw vpn-network-accesses
|  +--rw vpn-network-access* [id]
|     ...
+--rw maximum-routes
|  +--rw address-family* [af]
|     +--rw af
|     |       vpn-common:address-family
|     +--rw maximum-routes?   uint32
+--rw multicast {vpn-common:multicast}?
   ...
```

Figure 9: VPN Node Subtree Structure

7.3.4.1. RT/RD Assignment/auto-assignment

For the cases the logical resources are managed outside the network controller, the model allows to explicitely indicate the logical resources such as Route targets (RTs) and Route Distinguishers (RDs) (RT,RD).

Three possible behaviors are needed to fulfil the identified use cases:

- The network controller auto-assigns logical resources (RTs, RDs). This can apply for new services deployment.

- The Network Operator/Service orchestrator assigns explicitly the RTs and RDs. This case will fit with a brownfield scenario where some existing services needs to be updated by the network operators.

- The Network Operator/Service orchestrator explicitly wants NO RT/RD to be assigned. This case will fit in VRF-Lite scenarios, CE testing inside the Network or just for troubleshooting pruposes.

Thus a union between two yang data types are included in order to support this scenarios. So, if the leaf is not created in the Yang the expected behavior is the auto-assigns. If the Leaf is created with a valid rd value it will be explicitly assign in the VPN Node and if the leaf is created with an empty value, the RD value will not be deployed in the VPN node.

7.3.4.2. VPN Network Access

A 'vpn-network-access' represents an entry point to a VPN service (Figure 10). In other words, this container encloses the parameters that describe the access information for the traffic that belongs to a particular L3VPN. As such, every 'vpn-network-access' MUST belong to one and only one 'vpn-node'.

A 'vpn-network-access' includes information such as the connection on which the access is defined (see Section 7.3.4.2.1), the encapsulation of the traffic, policies that are applied on the access, etc.

Each 'vpn-network-access' SHOULD have a 'vpn-network-access-type' to select the type of network interface to be deployed in the devices. The available options are:

- Point-to-Point: The point-to-point type represent a direct connection between the end-points. It implies the controller must keep the association between a logical or physical interface on the device with the 'id' of the vpn-network-access.

- Multipoint: This option represents a broadcast connection between end-points. It implies the controller must keep the association between a logical or physical interface on the device with the 'id' of the 'vpn-network-access'.

- Pseudowire: Represent a connection coming from an L2VPN service. It implies the controller must keep the relationship between the logical tunnels or bridges on the devices with the 'id' of the' vpn-network-access'.

- Loopback: It represents the creation of a logical interface on the devices.

A PNC [RFC8453] will accept VPN requests containing this construct, using the enclosed data to: configure the router's interface to include the parameters described at the 'vpn-network-access', include the given interface into a VRF, configuring policies or schedulers for processing the incoming traffic, etc.

```
    ...
    +--rw vpn-services
       +--rw vpn-service* [vpn-id]
          +--rw vpn-id                  vpn-common:vpn-id
          +  ...
           +--rw vpn-node* [ne-id]
              +--rw ne-id                      string
              + ...
              +--rw vpn-network-accesses
              |  +--rw vpn-network-access* [id]
              |     +--rw id
              |     |       vpn-common:vpn-id
              |     +--rw port-id?
              |     |       vpn-common:vpn-id
              |     +--rw description?              string
              |     +--rw status
              |     |  +--rw admin-enabled?   boolean
              |     |  +--ro oper-status?     operational-type
              |     +--rw vpn-network-access-type?   identityref
              |     +--rw connection
              |     |  ...
              |     +--rw ip-connection
              |     |  ...
              |     +--rw security
              |     |  ...
              |     +--rw routing-protocols
              |     |  ...
              |     +--rw service
              |        ...
              ...
```

Figure 10: VPN Network Access Tree Structure

7.3.4.2.1. Connection

The definition of a L3VPN is commonly specified not only at the IP layer, but also requires to identify parameters at the Ethernet layer, such as encapsulation type (e.g., VLAN, QinQ, QinAny, VxLAN, etc.). The 'connection' container represents and groups the set of Layer 2 connectivity from where the traffic of the L3VPN in a particular VPN Network access is coming.

Ethernet encapsulation description is not supported in [RFC8299]. However, this parameters are mandatory to configure the PE interfaces. Thus, in the L3NM, these parameters uses the connection container inside the 'vpn-network-access'. This container defines protocols and parameters to enable connectivity at Layer 2.

```
    ...
    +--rw vpn-services
       +--rw vpn-service* [vpn-id]
          +--rw vpn-id                 vpn-common:vpn-id
          +  ...
           +--rw vpn-node* [ne-id]
              +--rw ne-id                      string
              + ...
              +--rw vpn-network-accesses
              |  +--rw vpn-network-access* [id]
              |     +--rw id
              |     |       vpn-common:vpn-id
              |     ...
              |     +--rw connection
              |     |  +--rw encapsulation-type?   identityref
              |     |  +--rw logical-interface
              |     |  |  +--rw peer-reference?   uint32
              |     |  +--rw tagged-interface
              |     |  |  +--rw type?                identityref
              |     |  |  +--rw dot1q-vlan-tagged
              |     |  |  |       {vpn-common:dot1q}?
              |     |  |  |  +--rw tag-type?   identityref
              |     |  |  |  +--rw cvlan-id?   uint16
              |     |  |  +--rw priority-tagged
              |     |  |  |  +--rw tag-type?   identityref
              |     |  |  +--rw qinq {vpn-common:qinq}?
              |     |  |  |  +--rw tag-type?   identityref
              |     |  |  |  +--rw svlan-id    uint16
              |     |  |  |  +--rw cvlan-id    uint16
              |     |  |  +--rw qinany {vpn-common:qinany}?
              |     |  |  |  +--rw tag-type?   identityref
              |     |  |  |  +--rw svlan-id    uint16
              |     |  |  +--rw vxlan {vpn-common:vxlan}?
              |     |  |     +--rw vni-id       uint32
              |     |  |     +--rw peer-mode?   identityref
              |     |  |     +--rw peer-list* [peer-ip]
              |     |  |        +--rw peer-ip    inet:ip-address
              |     |  +--rw bearer
              |     |     ...
              |     ...
```

Figure 11: Encapsulation Subtree Structure

Additionally, the 'bearer-reference' and the pseudowire termination are supported (see Figure 12). A site, as per [RFC4176] represents a VPN customer's location that is connected to the Service Provider network via a CE-PE link, which can access at least one VPN. The connection from the site to the Service Provider network is the

bearer. Every site is associated with a list of bearers. A bearer is the layer two connections with the site. In the module it is assumed that the bearer has been allocated by the Service Provider at the service orchestration step. The bearer is associated to a network element and a port. Hence, a bearer is just a bearer-reference to allow the translation between L3SM and L3NM.

```
...
+--rw vpn-network-accesses
|  +--rw vpn-network-access* [id]
|     +--rw id
|     |       vpn-common:vpn-id
|     ...
|     +--rw vpn-network-access-type?   identityref
|     +--rw connection
|     |  ...
|     |  +--rw bearer
|     |     +--rw bearer-reference?   string
|     |     |       {vpn-common:bearer-reference}?
|     |     +--rw pseudowire
|     |     |  +--rw vcid?      uint32
|     |     |  +--rw far-end?   union
|     |     +--rw vpls
|     |        +--rw vcid?      union
|     |        +--rw far-end?   union
|  ...
```

Figure 12: Bearer Subtree Structure

7.3.4.2.2. IP Connections

IP connection container (Figure 13) has the parameters of the 'vpn-network-access' addressing information. The address allocated in this container would represent the PE interface address configuration. The IP connection container is designed to support both IPv4 and IPv6. It also supports three IP address assignment modes: SLAAC [RFC7527], Provider DHCP, DHCP relay, and static addressing. Only one of them is enabled for a given service.

```
   ...
   +--rw vpn-network-accesses
   |  +--rw vpn-network-access* [id]
   |     +--rw id
   |     |       vpn-common:vpn-id
   |     ...
   |     +--rw vpn-network-access-type?   identityref
   |     +--rw connection
   |     |  ...
```

```
|     +--rw ip-connection
|     |  +--rw ipv4 {vpn-common:ipv4}?
|     |  |  +--rw address-allocation-type?
|     |  |  |       identityref
|     |  |  +--rw (allocation-type)?
|     |  |     +--:(provider-dhcp)
|     |  |     |  +--rw provider-address?
|     |  |     |  |       inet:ipv4-address
|     |  |     |  +--rw prefix-length?
|     |  |     |  |       uint8
|     |  |     |  +--rw (address-assign)?
|     |  |     |     +--:(number)
|     |  |     |     |  +--rw number-of-dynamic-address?
|     |  |     |     |          uint16
|     |  |     |     +--:(explicit)
|     |  |     |        +--rw customer-addresses
|     |  |     |           +--rw address-group*
|     |  |     |                   [group-id]
|     |  |     |              +--rw group-id
|     |  |     |              |       string
|     |  |     |              +--rw start-address?
|     |  |     |              |       inet:ipv4-address
|     |  |     |              +--rw end-address?
|     |  |     |                      inet:ipv4-address
|     |  |     +--:(dhcp-relay)
|     |  |     |  +--rw dr-provider-address?
|     |  |     |  |       inet:ipv4-address
|     |  |     |  +--rw dr-prefix-length?
|     |  |     |  |       uint8
|     |  |     |  +--rw customer-dhcp-servers
|     |  |     |     +--rw server-ip-address*
|     |  |     |             inet:ipv4-address
|     |  |     +--:(static-addresses)
|     |  |        ...
|     |  +--rw ipv6 {vpn-common:ipv6}?
|     |  |  +--rw address-allocation-type?
|     |  |  |       identityref
|     |  |  +--rw (allocation-type)?
|     |  |     +--:(provider-dhcp)
|     |  |     |  +--rw (provider-dhcp)?
|     |  |     |     +--:(provider-address)
|     |  |     |     |  +--rw provider-address?
|     |  |     |     |          inet:ipv6-address
|     |  |     |     +--:(prefix-length)
|     |  |     |     |  +--rw prefix-length?
|     |  |     |     |          uint8
|     |  |     |     +--:(address-assign)
|     |  |     |        +--rw (address-assign)?
```

```
|     |  |     |           +--:(number)
|     |  |     |           |  +--rw number-of-dynamic-address?
|     |  |     |           |          uint16
|     |  |     |           +--:(explicit)
|     |  |     |              +--rw customer-addresses
|     |  |     |                 +--rw address-group*
|     |  |     |                         [group-id]
|     |  |     |                    +--rw group-id
|     |  |     |                    |       string
|     |  |     |                    +--rw start-address?
|     |  |     |                    |       inet:ipv6-address
|     |  |     |                    +--rw end-address?
|     |  |     |                            inet:ipv6-address
|     |  |     +--:(dhcp-relay)
|     |  |     |  +--rw dr-provider-address?
|     |  |     |  |       inet:ipv6-address
|     |  |     |  +--rw dr-prefix-length?
|     |  |     |  |       uint8
|     |  |     |  +--rw customer-dhcp-servers
|     |  |     |     +--rw server-ip-address*
|     |  |     |             inet:ipv6-address
|     |  |     +--:(static-addresses)
|     |  |        ...
```

Figure 13: IP Connection Subtree Structure

In the case of the static addressing (Figure 14), the model supports the assignment of several IP addresses in the same 'vpn-network-access'. To identify which of the addresses is the primary address of a connection ,the 'primary-address' reference MUST be set with the corresponding 'address-id'.

```
            ...
            +--rw vpn-network-accesses
               |  +--rw vpn-network-access* [id]
               |     +--rw id
               |     |       vpn-common:vpn-id
               |     ...
               |     +--rw vpn-network-access-type?   identityref
               |     +--rw connection
               |     |  ...
               |     +--rw ip-connection
               |     |  +--rw ipv4 {vpn-common:ipv4}?
               |     |  |  +--rw address-allocation-type?
               |     |  |  |       identityref
               |     |  |  +--rw (allocation-type)?
               |     |  |     ...
               |     |  |     +--:(static-addresses)
               |     |  |        +--rw primary-address?
               |     |  |        |       -> ../address/address-id
               |     |  |        +--rw address* [address-id]
               |     |  |           +--rw address-id
               |     |  |           |       string
               |     |  |           +--rw s-provider-address?
               |     |  |           |       inet:ipv4-address
               |     |  |           +--rw s-customer-address?
               |     |  |           |       inet:ipv4-address
               |     |  |           +--rw s-prefix-length?
               |     |  |                   uint8
               |     |  +--rw ipv6 {vpn-common:ipv6}?
               |     |  |  +--rw address-allocation-type?
               |     |  |  |       identityref
               |     |  |  +--rw (allocation-type)?
               |     |  |     ...
               |     |  |     +--:(static-addresses)
               |     |  |        +--rw s-primary-address?
               |     |  |        |       -> ../s-address/address-id
               |     |  |        +--rw s-address* [address-id]
               |     |  |           +--rw address-id
               |     |  |           |       string
               |     |  |           +--rw provider-address?
               |     |  |           |       inet:ipv6-address
               |     |  |           +--rw customer-address?
               |     |  |           |       inet:ipv6-address
               |     |  |           +--rw prefix-length?     uint8
               |     ...
```

Figure 14: IP Connection Subtree Structure: Static Mode

7.3.4.2.3. Security

The 'security' container specifies the authentication and the encryption to be applied for a given VPN network access (Figure 15).

```
...
+--rw vpn-network-accesses
|  +--rw vpn-network-access* [id]
|     +--rw id
|     |       vpn-common:vpn-id
|     +  ...
|     +--rw connection
|     |  ...
|     +--rw ip-connection
|     |  ...
|     +--rw security
|     |  +--rw encryption {vpn-common:encryption}?
|     |  |  +--rw enabled?   boolean
|     |  |  +--rw layer?     enumeration
|     |  +--rw encryption-profile
|     |     +--rw (profile)?
|     |     |  +--:(provider-profile)
|     |     |  |  +--rw profile-name?    leafref
|     |     |  +--:(customer-profile)
|     |     |     +--rw algorithm?       string
|     |     +--rw (key-type)?
|     |        +--:(psk)
|     |           +--rw preshared-key?   string
|     +--rw routing-protocols
|     |  ...
|     +--rw service
|        ...
|  ...
```

Figure 15: Security Subtree Structure

7.3.4.2.4. CE-PE Routing Protocols

The model allows the Provider to configure one or more routing protocols associated with a particular 'vpn-network-access' (Figure 16). This protocol will run between the PE and the CE. A routing protocol instance MUST have a type (e.g., bgp, ospf) and an identifier. The identifier is necessary when multiple instances of the same protocol have to be configured.

```
...
+--rw vpn-network-accesses
|  +--rw vpn-network-access* [id]
|     +--rw id
|     |       vpn-common:vpn-id
|     ...
|     +--rw ip-connection
|     |  ...
|     +--rw routing-protocols
|     |  +--rw routing-protocol* [id]
|     |     +--rw id                  string
|     |     +--rw type?               identityref
|     |     +--rw routing-profiles* [id]
|     |     |  +--rw id      leafref
|     |     |  +--rw type?   identityref
|     |     +--rw ospf {vpn-common:rtg-ospf}?
|     |     |  ...
|     |     +--rw bgp {vpn-common:rtg-bgp}?
|     |     |  ...
|     |     +--rw isis {vpn-common:rtg-isis}?
|     |     |  ...
|     |     +--rw static
|     |     |  ...
|     |     +--rw rip {vpn-common:rtg-rip}?
|     |     |  +--rw address-family*
|     |     |          vpn-common:address-family
|     |     +--rw vrrp {vpn-common:rtg-vrrp}?
|     |        +--rw address-family*
|     |                vpn-common:address-family
|     +--rw service
|        ...
...
```

Figure 16: Routing Subtree Structure

Routing configuration does not include low-level policies. These policies are low level device configurations that must not be part of an abstracted model. A provider's internal policies (such as security filters) will be implemented as part of the device configuration but does not require any input from this model. Some policies like primary/backup or load-balancing can be inferred from 'access-priority'.

When configuring multiple instances of the same routing protocol, this does not automatically imply that, from a device configuration perspective, there will be parallel instances (multiple processes) running. It will be up to the implementation to use the most appropriate deployment model. As an example, when multiple BGP peers

need to be implemented, multiple instances of BGP must be configured as part of this model. However, from a device configuration point of view, this could be implemented as:

- Multiple BGP processes with a single neighbor running in each process.
- A single BGP process with multiple neighbors running.
- A combination of both.

To be aligned with [RFC8299], this model supports the following CE-PE routing protocols:

- OSPF: The model (Figure 17) allows the user to configure OSPF to run as routing protocol on the 'vpn-network-access interface'. An OSPF instance can be bound to IPv4, IPv6 or both. When only IPv4 address-family is requested, it will be up to the implementation to drive whether OSPFv2 or OSPFv3 is used.

```
...
+--rw vpn-network-accesses
|  +--rw vpn-network-access* [id]
|     +--rw id
|     |       vpn-common:vpn-id
|     ...
|     +--rw ip-connection
|     |  ...
|     +--rw routing-protocols
|     |  +--rw routing-protocol* [id]
|     |     +--rw id                  string
|     |     +--rw type?               identityref
|     |     +--rw routing-profiles* [id]
|     |     |  +--rw id      leafref
|     |     |  +--rw type?   identityref
|     |     +--rw ospf {vpn-common:rtg-ospf}?
|     |     |  +--rw address-family*
|     |     |  |       vpn-common:address-family
|     |     |  +--rw area-address
|     |     |  |       yang:dotted-quad
|     |     |  +--rw metric?           uint16
|     |     |  +--rw mtu?              uint16
|     |     |  +--rw process-id?       uint16
|     |     |  +--rw security
|     |     |  |  +--rw auth-key?   string
|     |     |  +--rw sham-links
|     |     |       {vpn-common:rtg-ospf-sham-link}?
|     |     |     +--rw sham-link* [target-site]
|     |     |        +--rw target-site
|     |     |        |       vpn-common:vpn-id
|     |     |        +--rw metric?        uint16
|     |     +--rw bgp {vpn-common:rtg-bgp}?
|     |     |  ...
|     |     +--rw isis {vpn-common:rtg-isis}?
|     |     |  ...
|     |     +--rw static
|     |     |  ...
|     |     +--rw rip {vpn-common:rtg-rip}?
|     |     |  ...
|     |     +--rw vrrp {vpn-common:rtg-vrrp}?
|     |        ...
|     +--rw service
|        ...
...
```

Figure 17: OPSF Routing Subtree Structure

- BGP: The model (Figure 18) allows to configure a BGP neighbor, including a set for parameters that are pertinent to be tweaked at the network level for service customization purposes. This container does not aim to include every BGP parameter; a comprehensive set of parameters belongs more to the BGP device model. The following parameters are captured in Figure 18. It is up to the implementation to drive the corresponding BGP device configuration.

  * 'peer-autonomous-system': This parameter conveys the Customer's AS Number (ASN).

  * 'local-autonomous-system': This parameter is set of AS override is activated for this peer.

  * 'address-family': This attribute indicates the address-family of the peer. It can be set to IPv4, IPv6, or both address-families.

  * 'neighbor': The module supports supplying two neighbors (each for a given address-family) or one neighbor (if 'address-family' attribute is set to both IPv4 and IPv6 address-families). A list of IP address(es) of the BGP neighbor can be then conveyed in this parameter.

  * 'multihop': This attribute indicates the number of allowed IP hops between a BGP peer and a PE.

  * 'security': The authentication type will be driven by the implementation but the module supports any authentication that uses a key as a parameter.

  * 'as-override': If set, this parameter indicates whether AS override is enabled, i.e., replace the ASN of the peer specified in the AS Path attribute with the ASN identified by the 'local-autonomous-system' attribute.

  * 'default-route': This attribute controls whether default route(s) can be advertised to the peer.

  * 'bgp-max-prefix': This attribute is used to control how many prefixes can be received from a neighbor. If reached, the BGP session will be teared down.

  * 'bgp-timer': Two timers can be captured in this container: (1) 'hold-time' which is the time interval that will be used for the HoldTimer (Section 4.2 of [RFC4271]) when establishing a BGP session. (2) 'keep-alive' which is the time interval for

the KeepAlive timer between a PE and a BGP peer (Section 4.4 of [RFC4271]).

```
...
+--rw vpn-network-accesses
|  +--rw vpn-network-access* [id]
|     +--rw id
|     |       vpn-common:vpn-id
|     ...
|     +--rw ip-connection
|     |  ...
|     +--rw routing-protocols
|     |  +--rw routing-protocol* [id]
|     |     +--rw id                  string
|     |     +--rw type?               identityref
|     |     +--rw routing-profiles* [id]
|     |     |  +--rw id      leafref
|     |     |  +--rw type?   identityref
|     |     +--rw ospf {vpn-common:rtg-ospf}?
|     |     |  ...
|     |     +--rw bgp {vpn-common:rtg-bgp}?
|     |     |  +--rw peer-autonomous-system
|     |     |  |           inet:as-number
|     |     |  +--rw local-autonomous-system?
|     |     |  |       inet:as-number
|     |     |  +--rw address-family*
|     |     |  |      vpn-common:address-family
|     |     |  +--rw neighbor*
|     |     |  |       inet:ip-address
|     |     |  +--rw multihop?            uint8
|     |     |  +--rw security
|     |     |  |  +--rw auth-key?   string
|     |     |  +--rw status
|     |     |  |  +--rw admin-status
|     |     |  |  |  +--rw status?       identityref
|     |     |  |  |  +--rw last-updated?
|     |     |  |  |  |            yang:date-and-time
|     |     |  |  +--ro oper-status
|     |     |  |     +--ro status?       identityref
|     |     |  |     +--ro last-updated?
|     |     |  |                  yang:date-and-time
|     |     |  +--rw description?             string
|     |     |  +--rw as-override?            boolean
|     |     |  +--rw default-route?          boolean
|     |     |  +--rw bgp-max-prefix
|     |     |  |  +--rw max-prefix?           uint32
|     |     |  |  +--rw warning-threshold? decimal64
|     |     |  |  +--rw violate-action?  enumeration
```

```
|     |     |  |  +--rw restart-interval?     uint16
|     |     |  +--rw bgp-timer
|     |     |     +--rw keep-alive?   uint16
|     |     |     +--rw hold-time?    uint16
|     |     +--rw isis {vpn-common:rtg-isis}?
|     |     |  ...
|     |     +--rw static
|     |     |  ...
|     |     +--rw rip {vpn-common:rtg-rip}?
|     |     |  ...
|     |     +--rw vrrp {vpn-common:rtg-vrrp}?
|     |        ...
|     +--rw service
|        ...
...
```

Figure 18: BGP Routing Subtree Structure

o IS-IS: The model (Figure 19) allows the user to configure IS-IS to run on the 'vpn-network-access' interface. An IS-IS instance can run L1, L2, or both levels.

```
...
+--rw vpn-network-accesses
|  +--rw vpn-network-access* [id]
|     +--rw id
|     |       vpn-common:vpn-id
|     ...
|     +--rw ip-connection
|     |  ...
|     +--rw routing-protocols
|     |  +--rw routing-protocol* [id]
|     |     +--rw id                  string
|     |     +--rw type?
|     |     |        identityref
|     |     +--rw routing-profiles* [id]
|     |     |  +--rw id      leafref
|     |     |  +--rw type?   identityref
|     |     +--rw ospf {vpn-common:rtg-ospf}?
|     |     |  ...
|     |     +--rw bgp {vpn-common:rtg-bgp}?
|     |     |  ...
|     |     +--rw isis {vpn-common:rtg-isis}?
|     |     |  +--rw address-family*
|     |     |  |       vpn-common:address-family
|     |     |  +--rw area-address
|     |     |  |       yang:dotted-quad
|     |     |  +--rw level?            identityref
```

```
|     |     |  +--rw metric?          uint16
|     |     |  +--rw process-id?      uint16
|     |     |  +--rw mode?            enumeration
|     |     |  +--rw status
|     |     |     +--rw admin-status
|     |     |     |  +--rw status?
|     |     |     |  |       identityref
|     |     |     |  +--rw last-updated?
|     |     |     |          yang:date-and-time
|     |     |     +--ro oper-status
|     |     |        +--ro status?
|     |     |        |       identityref
|     |     |        +--ro last-updated?
|     |     |                yang:date-and-time
|     |     +--rw static
|     |     |  ...
|     |     +--rw rip {vpn-common:rtg-rip}?
|     |     |  ...
|     |     +--rw vrrp {vpn-common:rtg-vrrp}?
|     |        ...
|     +--rw service
|        ...
...
```

Figure 19: IS-IS Routing Subtree Structure

- RIP: The module covers only a list of address-family as parameter.
- VRRP: The module covers only a list of address-family as parameter.

The module allows a user to configure one or more IPv4 and/or IPv6 static routes as depicted in Figure 20.

```
...
+--rw vpn-network-accesses
|  +--rw vpn-network-access* [id]
|     +--rw id
|     |       vpn-common:vpn-id
|     ...
|     +--rw ip-connection
|     |  ...
|     +--rw routing-protocols
|     |  +--rw routing-protocol* [id]
|     |     +--rw id              string
|     |     +--rw type?      identityref
|     |     +--rw routing-profiles* [id]
|     |     |  +--rw id      leafref
|     |     |  +--rw type?   identityref
|     |     +--rw ospf {vpn-common:rtg-ospf}?
|     |     |  ...
|     |     +--rw bgp {vpn-common:rtg-bgp}?
|     |     |  ...
|     |     +--rw isis {vpn-common:rtg-isis}?
|     |     |  ...
|     |     +--rw static
|     |     |  +--rw cascaded-lan-prefixes
|     |     |     +--rw ipv4-lan-prefixes*
|     |     |     |       [lan next-hop]
|     |     |     |       {vpn-common:ipv4}?
|     |     |     |  +--rw lan
|     |     |     |  |       inet:ipv4-prefix
|     |     |     |  +--rw lan-tag?    string
|     |     |     |  +--rw next-hop
|     |     |     |          inet:ipv4-address
|     |     |     +--rw ipv6-lan-prefixes*
|     |     |             [lan next-hop]
|     |     |             {vpn-common:ipv6}?
|     |     |        +--rw lan
|     |     |        |       inet:ipv6-prefix
|     |     |        +--rw lan-tag?    string
|     |     |        +--rw next-hop
|     |     |                inet:ipv6-address
|     |     +--rw rip {vpn-common:rtg-rip}?
|     |     |  ...
|     |     +--rw vrrp {vpn-common:rtg-vrrp}?
|     |        ...
|     +--rw service
|        ...
...
```

Figure 20: Static Routing Subtree Structure

7.3.4.2.5. Services

The 'services' container specifies the service parameters to apply for a given VPN network access (Figure 21).

```
...
+--rw vpn-network-accesses
|  +--rw vpn-network-access* [id]
|     +--rw id
|     |       vpn-common:vpn-id
|     ...
|     +--rw service
|        +--rw svc-input-bandwidth     uint64
|        +--rw svc-output-bandwidth    uint64
|        +--rw svc-mtu                 uint16
|        +--rw qos {vpn-common:qos}?
|        |  +--rw qos-classification-policy
|        |  |  +--rw rule* [id]
|        |  |     +--rw id
|        |  |     |       string
|        |  |     +--rw (match-type)?
|        |  |     |  +--:(match-flow)
|        |  |     |  |  +--rw (l3)?
|        |  |     |  |  |  +--:(ipv4)
|        |  |     |  |  |  |  ...
|        |  |     |  |  |  +--:(ipv6)
|        |  |     |  |  |     ...
|        |  |     |  |  +--rw (l4)?
|        |  |     |  |     +--:(tcp)
|        |  |     |  |     |  ...
|        |  |     |  |     +--:(udp)
|        |  |     |  |        ...
|        |  |     |  +--:(match-application)
|        |  |     |     +--rw match-application?
|        |  |     |             identityref
|        |  |     +--rw target-class-id?
|        |  |             string
|        |  +--rw qos-profile
|        |     +--rw qos-profile* [profile]
|        |        +--rw profile      leafref
|        |        +--rw direction?   identityref
|        +--rw carrierscarrier
|        |       {vpn-common:carrierscarrier}?
|        |  +--rw signalling-type?   enumeration
|        +--rw multicast {vpn-common:multicast}?
|           +--rw site-type?        enumeration
|           +--rw address-family?
|           |       vpn-common:address-family
|           +--rw protocol-type?    enumeration
|           +--rw remote-source?    boolean
...
```

Figure 21: Services Subtree Structure

The following attributes are defined:

- 'svc-input-bandwidth': Indicates the inbound bandwidth of the connection (i.e., download bandwidth from the SP to the site).

- 'svc-output-bandwidth': Indicates the outbound bandwidth of the connection (i.e., upload bandwidth from the site to the SP).

- 'svc-mtu': Indicates the MTU at service level. It can be the IP MTU or MPLS MTU, for example.

- 'carrierscarrier': Groups a set of parameters that are used when CsC is enabled such the use of BGP for signalling purposes [RFC8277].

- 'multicast': Specifies the multicast mode and other service-related attributes such as the address-family.

- 'qos': Is used to define QoS policies to apply on a given connection. Classification can be based on many criteria such as:

  * Layer 3: As shown in Figure 23, the model allow to classify based on any IP header field or a combination thereof. Both IPv4 and IPv6 are supported.

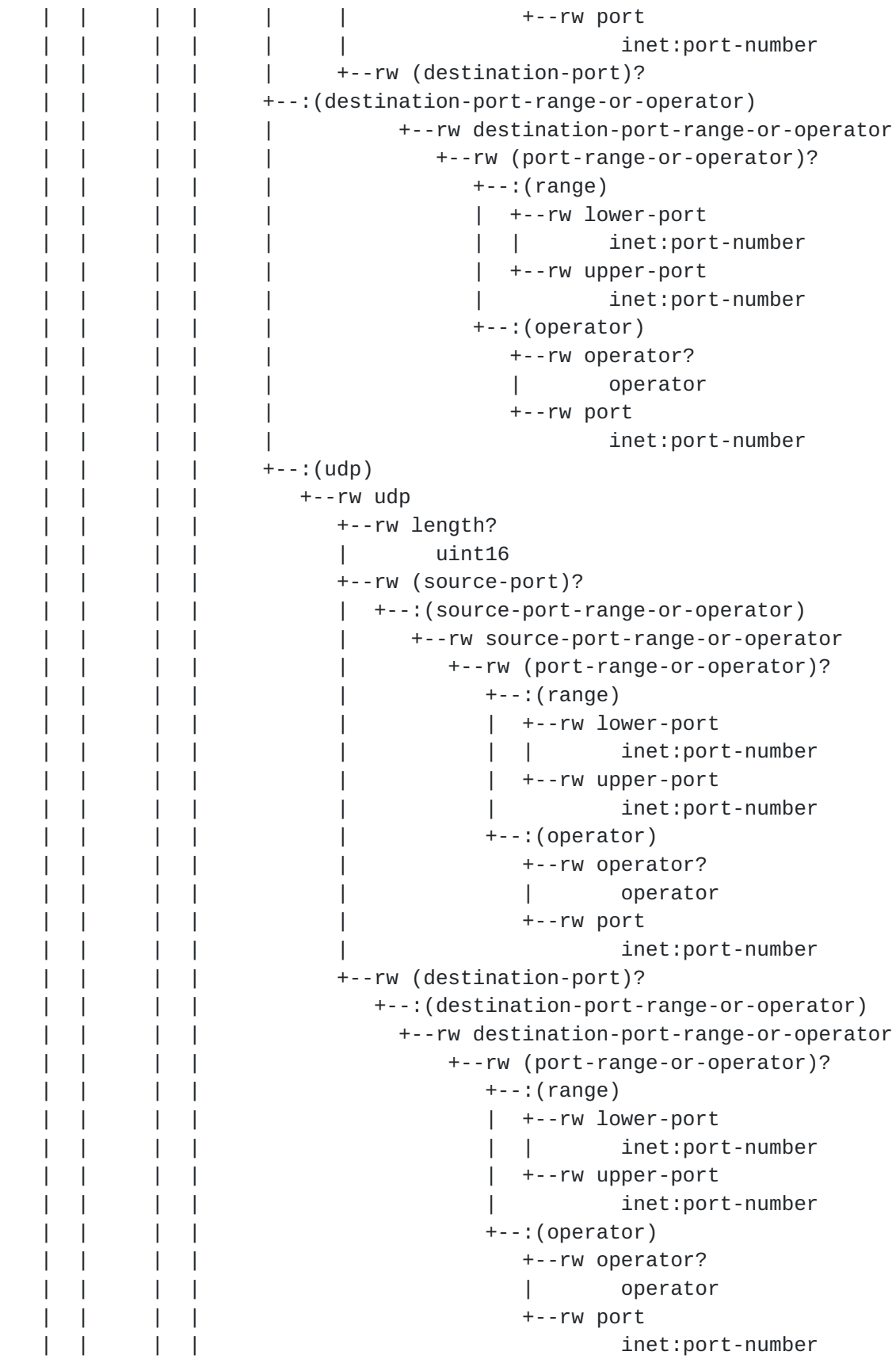
```
 +--rw qos {vpn-common:qos}?
 |  +--rw qos-classification-policy
 |  |  +--rw rule* [id]
 |  |     +--rw id
 |  |     |       string
 |  |     +--rw (match-type)?
 |  |     |  +--:(match-flow)
 |  |     |  |  +--rw (l3)?
 |  |     |  |  |  +--:(ipv4)
 |  |     |  |  |  |  ...
 |  |     |  |  |  +--:(ipv6)
 |  |     |  |  |     ...
 |  |     |  |  +--rw (l4)?
 |  |     |  |  +--rw (l3)?
 |  |     |  |  |  +--:(ipv4)
 |  |     |  |  |  |  +--rw ipv4
 |  |     |  |  |  |     +--rw dscp?
 |  |     |  |  |  |     |       inet:dscp
 |  |     |  |  |  |     +--rw ecn?
 |  |     |  |  |  |     |       uint8
 |  |     |  |  |  |     +--rw length?
 |  |     |  |  |  |     |       uint16
 |  |     |  |  |  |     +--rw ttl?
```

```
      |  |     |  |  |  |     |       uint8
      |  |     |  |  |  |     +--rw protocol?
      |  |     |  |  |  |     |       uint8
      |  |     |  |  |  |     +--rw ihl?
      |  |     |  |  |  |     |       uint8
      |  |     |  |  |  |     +--rw flags?
      |  |     |  |  |  |     |       bits
      |  |     |  |  |  |     +--rw offset?
      |  |     |  |  |  |     |       uint16
      |  |     |  |  |  |     +--rw identification?
      |  |     |  |  |  |     |       uint16
      |  |     |  |  |  |     +--rw (destination-network)?
      |  |     |  |  |  |     |  +--:(destination-ipv4-network)
      |  |     |  |  |  |     |     +--rw destination-ipv4-network?
      |  |     |  |  |  |     |             inet:ipv4-prefix
      |  |     |  |  |  |     +--rw (source-network)?
      |  |     |  |  |  |        +--:(source-ipv4-network)
      |  |     |  |  |  |           +--rw source-ipv4-network?
      |  |     |  |  |  |  inet:ipv4-prefix
      |  |     |  |  |  +--:(ipv6)
      |  |     |  |  |     +--rw ipv6
      |  |     |  |  |        +--rw dscp?
      |  |     |  |  |        |       inet:dscp
      |  |     |  |  |        +--rw ecn?
      |  |     |  |  |        |       uint8
      |  |     |  |  |        +--rw length?
      |  |     |  |  |        |       uint16
      |  |     |  |  |        +--rw ttl?
      |  |     |  |  |        |       uint8
      |  |     |  |  |        +--rw protocol?
      |  |     |  |  |        |       uint8
      |  |     |  |  |        +--rw (destination-network)?
      |  |     |  |  |        |  +--:(destination-ipv6-network)
      |  |     |  |  |        |     +--rw destination-ipv6-network?
      |  |     |  |  |        |             inet:ipv6-prefix
      |  |     |  |  |        +--rw (source-network)?
      |  |     |  |  |        |  +--:(source-ipv6-network)
      |  |     |  |  |        |     +--rw source-ipv6-network?
      |  |     |  |  |        |             inet:ipv6-prefix
      |  |     |  |  |        +--rw flow-label?
      |  |     |  |  |                inet:ipv6-flow-label
      |  |     |  |  +--rw (l4)?
      |  |     |  |     +--:(tcp)
      |  |     |  |     |  ...
      |  |     |  |     +--:(udp)
      |  |     |  |        ...
      ...
```

Figure 22: QoS Subtree Structure (L3)

* Layer 4: As shown in Figure 23, TCP or UDP-related match crietria can be specified.

```
+--rw qos {vpn-common:qos}?
|  +--rw qos-classification-policy
|  |  +--rw rule* [id]
|  |     +--rw id
|  |     |       string
|  |     +--rw (match-type)?
|  |     |  +--:(match-flow)
|  |     |  |  +--rw (l3)?
|  |     |  |  |  +--:(ipv4)
|  |     |  |  |  |  ...
|  |     |  |  |  +--:(ipv6)
|  |     |  |  |     ...
|  |     |  |  +--rw (l4)?
|  |     |  |     +--:(tcp)
|  |     |  |     |  +--rw tcp
|  |     |  |     |     +--rw sequence-number?
|  |     |  |     |     |       uint32
|  |     |  |     |     +--rw acknowledgement-number?
|  |     |  |     |     |       uint32
|  |     |  |     |     +--rw data-offset?
|  |     |  |     |     |       uint8
|  |     |  |     |     +--rw reserved?
|  |     |  |     |     |       uint8
|  |     |  |     |     +--rw flags?
|  |     |  |     |     |       bits
|  |     |  |     |     +--rw window-size?
|  |     |  |     |     |       uint16
|  |     |  |     |     +--rw urgent-pointer?
|  |     |  |     |     |       uint16
|  |     |  |     |     +--rw options?
|  |     |  |     |     |       binary
|  |     |  |     |     +--rw (source-port)?
|  |     |  |     |     |  +--:(source-port-range-or-operator)
|  |     |  |     |     |     +--rw source-port-range-or-operator
|  |     |  |     |     |        +--rw (port-range-or-operator)?
|  |     |  |     |     |           +--:(range)
|  |     |  |     |     |           |  +--rw lower-port
|  |     |  |     |     |           |  |       inet:port-number
|  |     |  |     |     |           |  +--rw upper-port
|  |     |  |     |     |           |          inet:port-number
|  |     |  |     |     |           +--:(operator)
|  |     |  |     |     |              +--rw operator?
|  |     |  |     |     |              |       operator
```


| | | | | | | +--rw port
| | | | | | | inet:port-number
| | | | | | +--rw (destination-port)?
| | | | | +--:(destination-port-range-or-operator)
| | | | | | +--rw destination-port-range-or-operator
| | | | | | +--rw (port-range-or-operator)?
| | | | | | +--:(range)
| | | | | | | +--rw lower-port
| | | | | | | | inet:port-number
| | | | | | | +--rw upper-port
| | | | | | | inet:port-number
| | | | | | +--:(operator)
| | | | | | +--rw operator?
| | | | | | | operator
| | | | | | +--rw port
| | | | | | inet:port-number
| | | | | +--:(udp)
| | | | | +--rw udp
| | | | | +--rw length?
| | | | | | uint16
| | | | | +--rw (source-port)?
| | | | | | +--:(source-port-range-or-operator)
| | | | | | +--rw source-port-range-or-operator
| | | | | | +--rw (port-range-or-operator)?
| | | | | | +--:(range)
| | | | | | | +--rw lower-port
| | | | | | | | inet:port-number
| | | | | | | +--rw upper-port
| | | | | | | inet:port-number
| | | | | | +--:(operator)
| | | | | | +--rw operator?
| | | | | | | operator
| | | | | | +--rw port
| | | | | | inet:port-number
| | | | | +--rw (destination-port)?
| | | | | +--:(destination-port-range-or-operator)
| | | | | +--rw destination-port-range-or-operator
| | | | | +--rw (port-range-or-operator)?
| | | | | +--:(range)
| | | | | | +--rw lower-port
| | | | | | | inet:port-number
| | | | | | +--rw upper-port
| | | | | | inet:port-number
| | | | | +--:(operator)
| | | | | +--rw operator?
| | | | | | operator
| | | | | +--rw port
| | | | | inet:port-number

```
                  ...
```

Figure 23: QoS Subtree Structure (L4)

* Application match

7.3.4.3. Multicast

Multicast MAY be enabled for a particular vpn-network-node (see Figure 24).

The model supports a single type of tree (Any-Source Multicast (ASM), Source-Specific Multicast (SSM), or bidirectional).

When ASM is used, the model supports the configuration of rendez-vous points (RPs). RP discovery may be 'static', 'bsr-rp', or 'auto-rp'. When set to 'static', RP to multicast grouping mapping MUST be configured as part of the 'rp-group-mappings' container. The RP MAY be a provider node or a customer node. When the RP is a customer node, the RP address must be configured using the 'rp-address' leaf otherwise no RP address is needed.

The model supports RP redundancy through the 'rp-redundancy' leaf. How the redundancy is achieved is out of scope and is up to the implementation.

When a particular VPN using ASM requires a more optimal traffic delivery, 'optimal-traffic-delivery' can be set. When set to 'true', the implementation must use any mechanism to provide a more optimal traffic delivery for the customer. Anycast is one of the mechanisms to enhance RPs redundancy, resilience against failures, and to recover from failures quickly.

For redundancy purposes, Multicast Source Discovery Protocol (MSDP) [RFC3618] may be enabled and used to share the state about sources between multiple RPs. The purpose of MSDP in this context is to enhance the robustness of the multicast service. MSDP may be configured on Non-RP routers, which is useful in a domain that does not support multicast sources, but does support multicast transit.

```
                  ...
                  +--rw vpn-network-accesses
                  |  +--rw vpn-network-access* [id]
                  |     +--rw id
                  |     ..
                  +--rw multicast {vpn-common:multicast}?
                     +--rw enabled?       boolean
                     +--rw tree-flavor*   identityref
```

```
                    +--rw rp
                    |  +--rw rp-group-mappings
                    |  |  +--rw rp-group-mapping* [id]
                    |  |     +--rw id                    uint16
                    |  |     +--rw provider-managed
                    |  |     |  +--rw enabled?
                    |  |     |  |       boolean
                    |  |     |  +--rw rp-redundancy?
                    |  |     |  |       boolean
                    |  |     |  +--rw optimal-traffic-delivery?
                    |  |     |  |       boolean
                    |  |     |  +--rw anycast
                    |  |     |     +--rw local-address?
                    |  |     |     |       inet:ip-address
                    |  |     |     +--rw rp-set-address*
                    |  |     |             inet:ip-address
                    |  |     +--rw rp-address
                    |  |     |       inet:ip-address
                    |  |     +--rw groups
                    |  |        +--rw group* [id]
                    |  |           +--rw id
                    |  |           |       uint16
                    |  |           +--rw (group-format)
                    |  |              +--:(group-prefix)
                    |  |              |  +--rw group-address?
                    |  |              |          inet:ip-prefix
                    |  |              +--:(startend)
                    |  |                 +--rw group-start?
                    |  |                 |       inet:ip-address
                    |  |                 +--rw group-end?
                    |  |                         inet:ip-address
                    |  +--rw rp-discovery
                    |     +--rw rp-discovery-type?   identityref
                    |     +--rw bsr-candidates
                    |        +--rw bsr-candidate-address*
                    |                inet:ip-address
                    +--rw msdp {msdp}?
                       +--rw enabled?         boolean
                       +--rw peer?            inet:ip-address
                       +--rw local-address?   inet:ip-address
```

Figure 24: Multicast Subtree Structure

8. Layer 3 Network Model

This module uses types defined in [RFC6991] and groupings defined in [RFC8519].

```
<CODE BEGINS>  file "ietf-l3vpn-ntw@2020-10-16.yang"
module ietf-l3vpn-ntw {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-l3vpn-ntw";
  prefix l3nm;

  import ietf-vpn-common {
    prefix vpn-common;
    reference
      "RFC UUUU: A Layer 2/3 VPN Common YANG Model";
  }
  import ietf-inet-types {
    prefix inet;
    reference
      "Section 4 of RFC 6991";
  }
  import ietf-yang-types {
    prefix yang;
    reference
      "Section 3 of RFC 6991";
  }
  import ietf-packet-fields {
    prefix pf;
    reference
      "RFC 8519: YANG Data Model for Network Access
                 Control Lists (ACLs)";
  }

  organization
    "IETF OPSA (Operations and Management Area) Working Group ";
  contact
    "WG Web:   <http://tools.ietf.org/wg/opsawg/>
     WG List:  <mailto:opsawg@ietf.org>
     Editor:   Samier Barguil
               <mailto:samier.barguilgiraldo.ext@telefonica.com>
     Editor:   Oscar Gonzalez de Dios
               <mailto:oscar.gonzalezdedios@telefonica.com>
     Editor:   Mohamed Boucadair
              <mailto:mohamed.boucadair@orange.com>
     Author:   Luis Angel Munoz
               <mailto:luis-angel.munoz@vodafone.com>
     Author:   Alejandro Aguado
               <mailto:alejandro.aguado_martin@nokia.com>
    ";
  description
    "This YANG module defines a generic network-oriented model
     for the configuration of Layer 3 Virtual Private Networks.
```

```
  revision 2020-10-16 {
    description
      "Initial revision.";
    reference
      "RFC XXXX: A Layer 3 VPN Network YANG Model";
  }

  /* Features */

  feature msdp {
    description
      "This feature indicates that Multicast Source
       Discovery Protocol (MSDP) capabilities are
       supported by the VPN.";
    reference
      "RFC 3618: Multicast Source Discovery Protocol (MSDP)";
  }

  /* Typedefs */

  typedef area-address {
    type string {
      pattern '[0-9A-Fa-f]{2}(\.[0-9A-Fa-f]{4}){0,6}';
    }
    description
      "This type defines the area address format.";
  }

  /* Identities */

  identity address-allocation-type {
    description
      "Base identity for address-allocation-type for
       PE-CE link.";
```

```
  }

  identity provider-dhcp {
    base address-allocation-type;
    description
      "The Provider's network provides a DHCP service
       to the customer.";
  }

  identity provider-dhcp-relay {
    base address-allocation-type;
    description
      "The Provider's network provides a DHCP relay service
       to the customer.";
  }

  identity provider-dhcp-slaac {
    base address-allocation-type;
    description
      "The Provider's network provides a DHCP service to
       the customer, as well as IPv6 Stateless Address
       Autoconfiguration (SLAAC).";
    reference
      "RFC 7527: IPv6 Stateless Address Autoconfiguration";
  }

  identity static-address {
    base address-allocation-type;
    description
      "The Provider-to-customer addressing is static.";
  }

  identity slaac {
    base address-allocation-type;
    description
      "Use IPv6 SLAAC.";
    reference
      "RFC 7527: IPv6 Stateless Address Autoconfiguration";
  }

  identity isis-level {
    description
      "Defines the IS-IS level for interface
       and system.";
  }

  identity level1 {
    base isis-level;
```

```
    description
      "IS-IS level 1.";
  }

  identity level2 {
    base isis-level;
    description
      "IS-IS level 2.";
  }

  identity level1-2 {
    base isis-level;
    description
      "IS-IS levels 1 and 2.";
  }

  identity bearer-inf-type {
    description
      "Identity for the bearer interface type.";
  }

  identity port-id {
    base bearer-inf-type;
    description
      "Identity for the priority-tagged interface.";
  }

  identity lag-id {
    base bearer-inf-type;
    description
      "Identity for the lag-tagged interface.";
  }

  /* Groupings */

  grouping security-params {
    container security {
      leaf auth-key {
        type string;
        description
          "MD5 authentication password for the connection
           towards the customer edge.";
      }
      description
        "Container for aggregating any security parameter
          for routing sessions between a PE and a CE.";
    }
    description
```

```
      "Grouping to define a set of security parameters";
  }

  grouping ports {
    choice source-port {
      container source-port-range-or-operator {
        uses pf:port-range-or-operator;
        description
          "Source port definition.";
      }
      description
        "Choice of specifying the source port or
          referring to a group of source port numbers.";
    }
    choice destination-port {
      container destination-port-range-or-operator {
        uses pf:port-range-or-operator;
        description
          "Destination port definition.";
      }
      description
        "Choice of specifying a destination port or
         referring to a group of destination port
         numbers.";
    }
    description
      "Choice of specifying a source or destination
       port numbers.";
  }

  /* Main Blocks */
  /* Main l3nm */

  container l3vpn-ntw {
    container vpn-profiles {
      uses vpn-common:vpn-profile-cfg;
      description
        "Contains a set of valid VPN Profiles to
         reference in the VPN service.";
    }
    container vpn-services {
      list vpn-service {
        key "vpn-id";
        uses vpn-common:service-status;
        uses vpn-common:vpn-description;
        leaf l3sm-vpn-id {
          type vpn-common:vpn-id;
          description
```

```
            "Pointer to the parent L3SM service,
             if any.";
        }
        leaf vpn-type {
          type identityref {
            base vpn-common:vpn-signaling-type;
          }
          description
            "Indicates the service type";
        }
        leaf vpn-service-topology {
          type identityref {
            base vpn-common:vpn-topology;
          }
          default "vpn-common:any-to-any";
          description
            "VPN service topology.";
        }
        container ie-profiles {
          list ie-profile {
            key "ie-profile-id";
            leaf ie-profile-id {
              type string;
              description
                "IE profile id.";
            }
            uses vpn-common:rt-rd;
            description
              "List for Imort/Export profile.";
          }
          description
            "Container for Import/Export profiles.";
        }
        uses vpn-common:svc-transport-encapsulation;
        container vpn-nodes {
          description
            "Container for VPN nodes.";
          list vpn-node {
            key "vpn-node-id";
            leaf vpn-node-id {
              type union {
                type vpn-common:vpn-id;
                type uint32;
              }
              description
                "Type STRING or NUMBER Service-Id.";
            }
            leaf local-autonomous-system {
```

```
            type inet:as-number;
            description
              "Provider's AS number in case the customer
               requests BGP routing.";
          }
          leaf description {
            type string;
            description
              "Textual description of the VPN node.";
          }
          leaf ne-id {
            type string;
            description
              "Unique identifier of the network element
               where the VPN node is deployed.";
          }
          leaf router-id {
            type inet:ip-address;
            description
              "The router-id information can be an IPv4
               or IPv6 address.";
          }
          leaf address-family {
            type vpn-common:address-family;
            description
              "The address family used for router-id
               information.";
          }
          leaf node-role {
            type identityref {
              base vpn-common:role;
            }
            default "vpn-common:any-to-any-role";
            description
              "Role of the VPN node in the IP VPN.";
          }
          uses vpn-common:rt-rd;
          uses vpn-common:service-status;
          leaf node-ie-profile {
            type leafref {
              path "/l3vpn-ntw/vpn-services/"
                 + "vpn-service/ie-profiles/"
                 + "ie-profile/ie-profile-id";
            }
            description
              "Node's Import/Export profile.";
          }
          uses vpn-common:vpn-node-group;
```

```
            container vpn-network-accesses {
              list vpn-network-access {
                key "id";
                leaf id {
                  type vpn-common:vpn-id;
                  description
                    "Identifier for the access.";
                }
                leaf port-id {
                  type vpn-common:vpn-id;
                  description
                    "Identifier for the network access.";
                }
                leaf description {
                  type string;
                  description
                    "Textual description of a network access.";
                }
                uses vpn-common:service-status;
                leaf vpn-network-access-type {
                  type identityref {
                    base vpn-common:site-network-access-type;
                  }
                  default "vpn-common:point-to-point";
                  description
                    "Describes the type of connection, e.g.,
                     point-to-point or multipoint.";
                }
                container connection {
                  leaf encapsulation-type {
                    type identityref {
                      base vpn-common:encapsulation-type;
                    }
                    default "vpn-common:untagged-int";
                    description
                      "Encapsulation type.  By default,
                       the encapsulation type is set to
                       'untagged'.";
                  }
                  container logical-interface {
                    leaf peer-reference {
                      type uint32;
                      description
                        "Specify the associated logical peer
                         interface";
                    }
                    description
                      "Reference of a logical interface
```

```
          type.";
        }
        container tagged-interface {
          leaf type {
            type identityref {
              base vpn-common:encapsulation-type;
            }
            default "vpn-common:priority-tagged";
            description
              "Tagged interface type.  By default,
               the type of the tagged interface is
               'priority-tagged'.";
          }
          container dot1q-vlan-tagged {
            when "derived-from-or-self(../type, "
               + "'vpn-common:dot1q')" {
              description
                "Only applies when the type of the
                 tagged interface is 'dot1q'.";
            }
            if-feature "vpn-common:dot1q";
            leaf tag-type {
              type identityref {
                base vpn-common:tag-type;
              }
              default "vpn-common:c-vlan";
              description
                "Tag type.  By default, the tag
                 type is 'c-vlan'.";
            }
            leaf cvlan-id {
              type uint16;
              description
                "VLAN identifier.";
            }
            description
              "Tagged interface.";
          }
          container priority-tagged {
            when "derived-from-or-self(../type, "
               + "'vpn-common:priority-tagged')" {
              description
                "Only applies when the type of the
                 tagged interface is
                 'priority-tagged'.";
            }
            leaf tag-type {
              type identityref {
```

```
        base vpn-common:tag-type;
      }
      default "vpn-common:c-vlan";
      description
        "Tag type.  By default, the tag
         type is 'c-vlan'.";
    }
    description
      "Priority tagged.";
  }
  container qinq {
    when "derived-from-or-self(../type, "
       + "'vpn-common:qinq')" {
      description
        "Only applies when the type of
         the tagged interface is 'qinq'.";
    }
    if-feature "vpn-common:qinq";
    leaf tag-type {
      type identityref {
        base vpn-common:tag-type;
      }
      default "vpn-common:c-s-vlan";
      description
        "Tag type.  By default, the tag
         type is 'c-s-vlan'.";
    }
    leaf svlan-id {
      type uint16;
      mandatory true;
      description
        "SVLAN identifier.";
    }
    leaf cvlan-id {
      type uint16;
      mandatory true;
      description
        "CVLAN identifier.";
    }
    description
      "QinQ.";
  }
  container qinany {
    when "derived-from-or-self(../type, "
       + "'vpn-common:qinany')" {
      description
        "Only applies when the type of the
         tagged interface is 'qinany'.";
```

```
        }
        if-feature "vpn-common:qinany";
        leaf tag-type {
          type identityref {
            base vpn-common:tag-type;
          }
          default "vpn-common:s-vlan";
          description
            "Tag type.  By default, the tag type
             is 's-vlan'.";
        }
        leaf svlan-id {
          type uint16;
          mandatory true;
          description
            "Service VLAN ID.";
        }
        description
          "Container for QinAny.";
      }
      container vxlan {
        when "derived-from-or-self(../type, "
           + "'vpn-common:vxlan')" {
          description
            "Only applies when the type of the
             tagged interface is 'vxlan'.";
        }
        if-feature "vpn-common:vxlan";
        leaf vni-id {
          type uint32;
          mandatory true;
          description
            "VXLAN Network Identifier (VNI).";
        }
        leaf peer-mode {
          type identityref {
            base vpn-common:vxlan-peer-mode;
          }
          default "vpn-common:static-mode";
          description
            "Specifies the VXLAN access mode.
             By default, the peer mode is set
             to 'static-mode'.";
        }
        list peer-list {
          key "peer-ip";
          leaf peer-ip {
            type inet:ip-address;
```

```
            description
              "Peer IP.";
          }
          description
            "List of peer IP addresses.";
        }
        description
          "QinQ.";
      }
      description
        "Container for tagged interfaces.";
    }
    container bearer {
      leaf bearer-reference {
        if-feature "vpn-common:bearer-reference";
        type string;
        description
          "This is an internal reference for*
           the SP.";
      }
      container pseudowire {
        leaf vcid {
          type uint32;
          description
            "PW or VC identifier.";
        }
        leaf far-end {
          type union {
            type uint32;
            type inet:ipv4-address;
          }
          description
            "SDP/Far End/LDP neighbour reference.";
        }
        description
          "Pseudowire termination parameters";
      }
      container vpls {
        leaf vcid {
          type union {
            type uint32;
            type string;
          }
          description
            "VCID identifier, IRB/RVPPLs interface
             supported using string
             format.";
        }
```

```
          leaf far-end {
            type union {
              type uint32;
              type inet:ipv4-address;
            }
            description
              "SDP/Far End/LDP Neighbour reference.";
          }
          description
            "Pseudowire termination parameters";
        }
        description
          "Defines physical properties of a site
           attachment.";
      }
      description
        "Encapsulation types";
    }
    container ip-connection {
      container ipv4 {
        if-feature "vpn-common:ipv4";
        leaf address-allocation-type {
          type identityref {
            base address-allocation-type;
          }
          must "not(derived-from-or-self(current(), "
            + "'slaac') or derived-from-or-self(current(),"
            + " 'provider-dhcp-slaac'))" {
            error-message "SLAAC is only applicable to
                           IPv6";
          }
          description
            "Defines how addresses are allocated.
             If there is no value for the address
             allocation type, then IPv4 is not enabled.";
        }
        choice allocation-type {
          case provider-dhcp {
            when "derived-from-or-self(./address-"
               + "allocation-type, 'provider-dhcp')" {
              description
                "Only applies when addresses are
                 allocated by DHCP.";
            }
            leaf provider-address {
              type inet:ipv4-address;
              description
                "Address of provider side.
```

```
          If provider-address is not specified,
          then prefix length should not be
          specified either.

          It also implies provider-dhcp
          allocation is not enabled.

          If provider-address is specified,
          then the prefix length may or
          may not be specified.";
        }
        leaf prefix-length {
          type uint8 {
            range "0..32";
          }
          must '(../provider-address)' {
            error-message
              "If the prefix length is specified,
               provider-address must also be
               specified.";
            description
              "If the prefix length is specified,
               provider-address must also be
               specified.";
          }
          description
            "Subnet prefix length expressed in bits.
             If not specified, or specified as zero,
             this means the customer leaves the actual
             prefix length value to the provider.";
        }
        choice address-assign {
          default "number";
          case number {
            leaf number-of-dynamic-address {
              type uint16;
              default "1";
              description
                "Describes the number of IP
                 addresses the customer requires.";
            }
          }
          case explicit {
            container customer-addresses {
              list address-group {
                key "group-id";
                leaf group-id {
                  type string;
```

```
                    description
                      "Group-id for the address range
                       from start-address to
                       end-address.";
                  }
                  leaf start-address {
                    type inet:ipv4-address;
                    description
                      "First address.";
                  }
                  leaf end-address {
                    type inet:ipv4-address;
                    description
                      "Last address.";
                  }
                  description
                    "Describes IP addresses allocated by
                     DHCP.

                     When only start-address or only
                     end-address is present, it
                     represents a single address.
                     When both start-address and
                     end-address are specified, it
                     implies a range inclusive of
                     both addresses.  If no address
                     is specified, it implies customer
                     addresses group is not supported.";
                }
                description
                  "Container for customer addresses is
                   allocated by DHCP.";
              }
            }
            description
              "Choice for the way to assign
               addresses.";
          }
          description
            "DHCP allocated addresses related
             parameters.";
        }
        case dhcp-relay {
          when "derived-from-or-self(./address-allocation"
             + "-type, 'provider-dhcp-relay')" {
            description
              "Only applies when provider is required to
               implement DHCP relay function.";
```

```
                    }
                    leaf dr-provider-address {
                      type inet:ipv4-address;
                      description
                        "Address of provider side.

                         If provider-address is
                         not specified, then prefix length
                         should not be specified either.

                         It also implies provider-dhcp
                         allocation is not enabled.

                         If provider-address is specified,
                         then prefix length may or may
                         not be specified.";
                    }
                    leaf dr-prefix-length {
                      type uint8 {
                        range "0..32";
                      }
                      must '(../dr-provider-address)' {
                        error-message
                          "If prefix length is specified,
                           provider-address must also be
                           specified.";
                        description
                          "If prefix length is specified,
                           provider-address must also be
                           specified.";
                      }
                      description
                        "Subnet prefix length expressed in bits.

                         If not specified, or specified as zero,
                         this means the customer leaves the
                         actual prefix  length value
                         to the provider.";
                    }
                    container customer-dhcp-servers {
                      leaf-list server-ip-address {
                        type inet:ipv4-address;
                        description
                          "IP address of customer DHCP
                           server.";
                      }
                      description
                        "Container for list of customer
```

```
            DHCP servers.";
        }
        description
          "DHCP relay provided by operator.";
      }
      case static-addresses {
        when "derived-from-or-self(./address-allocation"
           + "-type, 'static-address')" {
          description
            "Only applies when address allocation
            type is static.";
        }
        leaf primary-address {
          type leafref {
            path "../address/address-id";
          }
          description
            "Principal address of the connection.";
        }
        list address {
          key "address-id";
          leaf address-id {
            type string;
            description
              "IPv4 Address";
          }
          leaf s-provider-address {
            type inet:ipv4-address;
            description
              "IPv4 Address List of the provider side.
               When the protocol allocation type is
               static, the provider address must be
               configured.";
          }
          leaf s-customer-address {
            type inet:ipv4-address;
            description
              "IPv4 Address of customer side.";
          }
          leaf s-prefix-length {
            type uint8 {
              range "0..32";
            }
            description
              "Subnet prefix length expressed
               in bits. It is applied to both
               provider-address and customer-address.";
          }
```

```
            description
              "Describes IPv4 addresses used.";
          }
          description
            "Describes IPv4 addresses used.";
        }
        description
          "Choice the address allocation.";
      }
      description
        "IPv4-specific parameters.";
    }
    container ipv6 {
      if-feature "vpn-common:ipv6";
      leaf address-allocation-type {
        type identityref {
          base address-allocation-type;
        }
        description
          "Defines how addresses are allocated.
           If there is no value for the address
           allocation type, then IPv6 is
           not enabled.";
      }
      choice allocation-type {
        choice provider-dhcp {
          when "derived-from-or-self(./address-allo"
             + "cation-type, 'provider-dhcp') "
             + "or derived-from-or-self(./address-allo"
             + "cation-type, 'provider-dhcp-slaac')" {
            description
              "Only applies when addresses are
               allocated by DHCP.";
          }
          leaf provider-address {
            type inet:ipv6-address;
            description
              "Address of the provider side.

               If provider-address is not specified,
               then prefix length should not be
               specified either.  It also implies
               provider-dhcp allocation is not
               enabled.

               If provider-address is
               specified, then prefix length may
               or may not be specified.";
```

```
                    }
                    leaf prefix-length {
                      type uint8 {
                        range "0..128";
                      }
                      must '(../provider-address)' {
                        error-message
                          "If prefix length is specified,
                           provider-address
                           must also be specified.";
                        description
                          "If prefix length is specified,
                            provider-address
                           must also be specified.";
                      }
                      description
                        "Subnet prefix length expressed in
                         bits.

                         If not specified, or specified as
                         zero, this means the customer leaves
                         the actual prefix length value to
                         the provider.";
                    }
                    choice address-assign {
                      default "number";
                      case number {
                        leaf number-of-dynamic-address {
                          type uint16;
                          default "1";
                          description
                            "Describes the number of IP
                             addresses required by the
                             customer.";
                        }
                      }
                      case explicit {
                        container customer-addresses {
                          list address-group {
                            key "group-id";
                            leaf group-id {
                              type string;
                              description
                                "Group-id for the address range
                                 from start-address to
                                 end-address.";
                            }
                            leaf start-address {
```

```
                  type inet:ipv6-address;
                  description
                    "First address.";
                }
                leaf end-address {
                  type inet:ipv6-address;
                  description
                    "Last address.";
                }
                description
                  "Describes IP addresses allocated
                   by DHCP.

                   When only start-address or only
                   end-address is present, it
                   represents a single address.

                   When both start-address and
                   end-address are specified, it
                   implies a range inclusive of
                   both addresses.

                   If no address is specified, it
                   implies customer addresses group
                   is not supported.";
              }
              description
                "Container for customer addresses
                 allocated by DHCP.";
            }
          }
          description
            "Choice for the way to assign addresses.";
        }
        description
          "DHCP allocated addresses related
           parameters.";
      }
      case dhcp-relay {
        when "derived-from-or-self(./address-allo"
           + "cation-type, 'provider-dhcp-relay')" {
          description
            "Only applies when the provider is required
             to implement DHCP relay function.";
        }
        leaf dr-provider-address {
          type inet:ipv6-address;
          description
```

```
        "Address of the provider side.

         If provider-address is not specified,
         then prefix length should not be
         specified either.  It also implies
         provider-dhcp allocation is not enabled.

         If provider address is specified, then
         prefix length may or may not be
         specified.";
      }
      leaf dr-prefix-length {
        type uint8 {
          range "0..128";
        }
        must '(../dr-provider-address)' {
          error-message
            "If prefix length is specified,
             provider-address must also be
             specified.";
          description
            "If prefix length is specified,
             provider-address must also be
             specified.";
        }
        description
          "Subnet prefix length expressed in bits.

           If not specified, or specified as zero,
           this means the customer leaves the
           actual prefix length value to the
           provider.";
      }
      container customer-dhcp-servers {
        leaf-list server-ip-address {
          type inet:ipv6-address;
          description
            "This node contains the IP address of
             the customer DHCP server.  If the DHCP
             relay function is implemented by the
             provider, this node contains the
             configured value.";
        }
        description
          "Container for list of customer DHCP
           servers.";
      }
      description
```

```
        "DHCP relay provided by operator.";
      }
      case static-addresses {
        when "derived-from-or-self(./address-allocation"
           + "-type, 'static-address')" {
          description
            "Only applies when protocol allocation type
             is static.";
        }
        leaf s-primary-address {
          type leafref {
            path "../s-address/address-id";
          }
          description
            "Principal address of the connection";
        }
        list s-address {
          key "address-id";
          leaf address-id {
            type string;
            description
              "IPv4 Address";
          }
          leaf provider-address {
            type inet:ipv6-address;
            description
              "IPv6 Address of the provider side.  When
               the protocol allocation type is static,
               the provider address must be
               configured.";
          }
          leaf customer-address {
            type inet:ipv6-address;
            description
              "The IPv6 Address of the customer side.";
          }
          leaf prefix-length {
            type uint8 {
              range "0..128";
            }
            description
              "Subnet prefix length expressed in bits.
               It is applied to both provider-address
               and customer-address.";
          }
          description
            "Describes IPv6 addresses used.";
        }
```

```
          description
            "IPv6-specific parameters.";
        }
        description
          "IPv6 allocation type.";
      }
      description
        "IPv6-specific parameters.";
    }
    container oam {
      container bfd {
        if-feature "vpn-common:bfd";
        leaf enabled {
          type boolean;
          default "false";
          description
            "If true, BFD activation is required.";
        }
        choice holdtime {
          default "fixed";
          case fixed {
            leaf fixed-value {
              type uint32;
              units "msec";
              description
                "Expected BFD holdtime.

                 The customer may impose some fixed
                 values for the holdtime period if the
                 provider allows the customer use this
                 function.

                 If the provider doesn't allow the
                 customer to use this function,
                 the fixed-value will not be set.";
            }
          }
          case profile {
            leaf profile-name {
              type leafref {
                path "/l3vpn-ntw/vpn-profiles/"
                   + "valid-provider-identifiers/"
                   + "bfd-profile-identifier/id";
              }
              description
                "Well-known SP profile name.

                 The provider can propose some profiles
```

```
                        to the customer, depending on the
                        service level the customer wants to
                        achieve.

                        Profile names must be communicated to
                        the customer.";
                    }
                    description
                      "Well-known SP profile.";
                  }
                  description
                    "Choice for holdtime flavor.";
                }
                description
                  "Container for BFD.";
              }
              description
                "Defines the Operations, Administration,
                 and Maintenance (OAM)mechanisms used on
                 the connection.

                 BFD is set as a fault detection mechanism,
                 but the 'oam' container can easily
                 be augmented by other mechanisms";
            }
            description
              "Defines connection parameters.";
          }
          container security {
            container encryption {
              if-feature "vpn-common:encryption";
              leaf enabled {
                type boolean;
                default "false";
                description
                  "If true, traffic encryption on the
                   connection is required. It is
                   disabled, otherwise.";
              }
              leaf layer {
                when "../enabled = 'true'" {
                  description
                    "Require a value for layer when
                     enabled is true.";
                }
                type enumeration {
                  enum layer2 {
                    description
```

```
                    "Encryption will occur at Layer 2.";
                }
                enum layer3 {
                  description
                    "Encryption will occur at Layer 3.
                     For example, IPsec may be used when
                     a customer requests Layer 3
                     encryption.";
                }
              }
              description
                "Layer on which encryption is applied.";
            }
            description
              "Container for CE-PE security encryption.";
          }
          container encryption-profile {
            choice profile {
              case provider-profile {
                leaf profile-name {
                  type leafref {
                    path "/l3vpn-ntw/vpn-profiles"
                       + "/valid-provider-identifiers"
                       + "/encryption-profile-identifier/id";
                  }
                  description
                    "Name of the SP profile to be applied.";
                }
              }
              case customer-profile {
                leaf algorithm {
                  type string;
                  description
                    "Encryption algorithm to be used.";
                }
              }
              description
                "Choice for encryption profile.";
            }
            choice key-type {
              default "psk";
              case psk {
                leaf preshared-key {
                  type string;
                  description
                    "Pre-Shared Key (PSK) coming from the
                     customer.";
                }
```

```
        }
        description
          "Choice of encryption profile.
           The encryption profile can be the
           provider profile or customer profile.";
      }
      description
        "Container for encryption profile.";
    }
    description
      "Site-specific security parameters.";
  }
  container routing-protocols {
    list routing-protocol {
      key "id";
      leaf id {
        type string;
        description
          "Unique identifier for routing protocol.";
      }
      leaf type {
        type identityref {
          base vpn-common:routing-protocol-type;
        }
        description
          "Type of routing protocol.";
      }
      list routing-profiles {
        key "id";
        leaf id {
          type leafref {
            path "/l3vpn-ntw/vpn-profiles"
               + "/valid-provider-identifiers"
               + "/routing-profile-identifier/id";
          }
          description
            "Routing profile to be used.";
        }
        leaf type {
          type identityref {
            base vpn-common:ie-type;
          }
          description
            "Import, export or both.";
        }
        description
          "Routing profiles.";
      }
```

```
                container ospf {
                  when "derived-from-or-self(../type, "
                     + "'vpn-common:ospf')" {
                    description
                      "Only applies when protocol is OSPF.";
                  }
                  if-feature "vpn-common:rtg-ospf";
                  leaf-list address-family {
                    type vpn-common:address-family;
                    min-elements 1;
                    description
                      "If OSPF is used on this site, this node
                       contains a configured value.  This node
                       contains at least one address family
                       to be activated.";
                  }
                  leaf area-address {
                    type yang:dotted-quad;
                    mandatory true;
                    description
                      "Area address.";
                  }
                  leaf metric {
                    type uint16;
                    default "1";
                    description
                      "Metric of the PE-CE link.  It is used
                       in the routing state calculation and
                       path selection.";
                  }
                  leaf mtu {
                    type uint16;
                    description
                      "Maximum transmission unit for a given
                       OSPF link.";
                  }
                  leaf process-id {
                    type uint16;
                    description
                      "Process id of the OSPF CE-PE connection.";
                  }
                  uses security-params;
                  container sham-links {
                    if-feature "vpn-common:rtg-ospf-sham-link";
                    list sham-link {
                      key "target-site";
                      leaf target-site {
                        type vpn-common:vpn-id;
```

```
            description
              "Target site for the sham link connection.
               The site is referred to by its ID.";
            }
            leaf metric {
              type uint16;
              default "1";
              description
                "Metric of the sham link.  It is used in
                 the routing state calculation and path
                 selection.  The default value is set
                 to 1.";
            }
            description
              "Creates a sham link with another site.";
          }
          description
            "List of sham links.";
        }
        description
          "OSPF-specific configuration.";
      }
      container bgp {
        when "derived-from-or-self(../type, "
           + "'vpn-common:bgp')" {
          description
            "Only applies when protocol is BGP.";
        }
        if-feature "vpn-common:rtg-bgp";
        leaf peer-autonomous-system {
          type inet:as-number;
          mandatory true;
          description
            "Indicates the Customer's AS Number (ASN) in
             case the Customer requests BGP routing.";
        }
        leaf local-autonomous-system {
          type inet:as-number;
          description
            "Is set to the ASN to override a peers' ASN
             if such feature is requested by the
             Customer.";
        }
        leaf-list address-family {
          type vpn-common:address-family;
          min-elements 1;
          description
            "This node contains at least one
```

```
        address-family to be activated.";
    }
    leaf-list neighbor {
      type inet:ip-address;
      description
        "IP address(es) of the BGP neighbor. IPv4
         and IPv6 neighbors may be indicated if
         two sessions will be used for IPv4 and
         IPv6.";
    }
    leaf multihop {
      type uint8;
      description
        "Describes the number of IP hops allowed
         between a given BGP neighbor and the PE.";
    }
    uses security-params;
    uses vpn-common:service-status;
    leaf description {
      type string;
      description
        "Includes a description of the BGP session.
         Such description is meant to be used for
         diagnosis purposes. The semantic of the
         description is local to an
         implementation.";
    }
    leaf as-override {
      type boolean;
      default "false";
      description
        "Defines whether AS override is enabled,
         i.e., replace the ASN of the peer specified
         in the AS Path attribute with the local
         AS number.";
    }
    leaf default-route {
      type boolean;
      default "false";
      description
        "Defines whether default route(s) can be
         advertised to its peer. If set, the
         default route(s) is advertised to its
         peer.";
    }
    container bgp-max-prefix {
      leaf max-prefix {
        type uint32;
```

```
        default "5000";
        description
          "Indicates the maximimum number of BGP
           prefixes allowed in the BGP session.

           It allows to control how many prefixes
           can be received from a neighbor.

           If the limit is exceeded, the session
            can be teared down.";
        reference
          "RFC4271, Section 8.2.2.";
      }
      leaf warning-threshold {
        type decimal64 {
          fraction-digits 5;
          range "0..100";
        }
        units "percent";
        default "75";
        description
          "When this value is reached, a warning
           notification will be triggered.";
      }
      leaf violate-action {
        type enumeration {
          enum warning {
            description
              "Only a warning message is sent to
               the peer when the limit is
               exceeded.";
          }
          enum discard-extra-paths {
            description
              "Discards extra paths when the
               limit is exceeded.";
          }
          enum restart {
            description
              "Restart after a time interval.";
          }
        }
        description
          "BGP neighbour max-prefix violate
           action";
      }
      leaf restart-interval {
        type uint16;
```

```
        units "minutes";
        description
          "Time interval (min) after which the
           BGP session will be reestablished.";
      }
      description
        "Controls the behavior when a prefix
         maximum is reached.";
    }
    container bgp-timer {
      description
        "Includes two BGP timers that can be
         customized when building a VPN service
         with BGP used as CE-PE routing
         protocol.";
      leaf keep-alive {
        type uint16 {
          range "0..21845";
        }
        units "seconds";
        default "30";
        description
          "This timer indicates the KEEPALIVE
           messages'  frequency between a PE
           and a BGP peer.

           If set to '0', it indicates KEEPALIVE
           messages are disabled.

           It is suggested that the maximum time
           between  KEEPALIVEmessages would be
           one third of the Hold Time interval.";
        reference
          "Section 4.4 of RFC 4271";
      }
      leaf hold-time {
        type uint16 {
          range "0 | 3..65535";
        }
        units "seconds";
        default "90";
        description
          "It indicates the maximum number of
          seconds that may elapse between the
          receipt of successive KEEPALIVE
          and/or UPDATE   messages from the peer.

          The Hold Time must be either zero or
```

```
            at least three seconds.";
          reference
            "Section 4.2 of RFC 4271";
        }
      }
      description
        "BGP-specific configuration.";
    }
    container isis {
      when "derived-from-or-self(../type, "
         + "'vpn-common:isis')" {
        description
          "Only applies when protocol is IS-IS.";
      }
      if-feature "vpn-common:rtg-isis";
      leaf-list address-family {
        type vpn-common:address-family;
        min-elements 1;
        description
          "If ISIS is used on this site, this node
           contains a configured value.  This node
           contains at least one address family
           to be activated.";
      }
      leaf area-address {
        type yang:dotted-quad;
        mandatory true;
        description
          "Area address.";
      }
      leaf level {
        type identityref {
          base isis-level;
        }
        description
          "level1, level2 or level1-2";
      }
      leaf metric {
        type uint16;
        default "1";
        description
          "Metric of the PE-CE link.  It is used
           in the routing state calculation and
           path selection.";
      }
      leaf process-id {
        type uint16;
        description
```

```
            "Process id of the IS-IS CE-PE
             connection.";
          }
          leaf mode {
            type enumeration {
              enum active {
                description
                  "Interface sends or receives IS-IS
                   protocol control packets.";
              }
              enum passive {
                description
                  "Suppresses the sending of IS-IS
                   updates through the specified
                   interface.";
              }
            }
            default "active";
            description
              "IS-IS interface mode type.";
          }
          uses vpn-common:service-status;
          description
            "IS-IS specific configuration.";
        }
        container static {
          when "derived-from-or-self(../type, "
             + "'vpn-common:static')" {
            description
              "Only applies when protocol is static.
               BGP activation requires the SP to know
               the address of the customer peer.  When
               BGP is enabled, the 'static-address'
               allocation type for the IP connection
               must be used.";
          }
          container cascaded-lan-prefixes {
            list ipv4-lan-prefixes {
              if-feature "vpn-common:ipv4";
              key "lan next-hop";
              leaf lan {
                type inet:ipv4-prefix;
                description
                  "LAN prefixes.";
              }
              leaf lan-tag {
                type string;
                description
```

```
                    "Internal tag to be used in VPN
                     policies.";
                  }
                  leaf next-hop {
                    type inet:ipv4-address;
                    description
                      "Next-hop address to use on the
                       customer side.";
                  }
                  description
                    "List of LAN prefixes for the site.";
                }
                list ipv6-lan-prefixes {
                  if-feature "vpn-common:ipv6";
                  key "lan next-hop";
                  leaf lan {
                    type inet:ipv6-prefix;
                    description
                      "LAN prefixes.";
                  }
                  leaf lan-tag {
                    type string;
                    description
                      "Internal tag to be used in VPN
                       policies.";
                  }
                  leaf next-hop {
                    type inet:ipv6-address;
                    description
                      "Next-hop address to use on the
                       customer side.";
                  }
                  description
                    "List of LAN prefixes for the site.";
                }
                description
                  "LAN prefixes from the customer.";
              }
              description
                "Configuration specific to static routing.";
            }
            container rip {
              when "derived-from-or-self(../type, "
                 + "'vpn-common:rip')" {
              description
                "Only applies when the protocol is RIP.
                 For IPv4, the model assumes that RIP
                 version 2 is used.";
```

```
      }
      if-feature "vpn-common:rtg-rip";
      leaf-list address-family {
        type vpn-common:address-family;
        min-elements 1;
        description
          "If RIP is used on this site, this node
           contains a configured value.  This node
           contains at least one address family
           to be activated.";
      }
      description
        "Configuration specific to RIP routing.";
    }
    container vrrp {
      when "derived-from-or-self(../type, "
         + "'vpn-common:vrrp')" {
        description
          "Only applies when protocol is VRRP.";
      }
      if-feature "vpn-common:rtg-vrrp";
      leaf-list address-family {
        type vpn-common:address-family;
        min-elements 1;
        description
          "If VRRP is used on this site, this node
           contains a configured value.  This node
           contains at least one address family to
           be activated.";
      }
      leaf vrrp-group {
        type uint8 {
          range "1..255";
        }
        description
          "VRRP group number";
      }
      leaf backup-peer {
        type inet:ip-address;
        description
          "IP address of the peer";
      }
      leaf priority {
        type uint8;
        description
          "Local priority of the VRRP speaker";
      }
      leaf ping-reply {
```

```
            type boolean;
            description
              "Whether the VRRP speaker should answer
               to ping requests";
          }
          description
            "Configuration specific to VRRP.";
        }
        description
          "List of routing protocols used on
           the site.  This list can be augmented.";
      }
      description
        "Defines routing protocols.";
    }
    container service {
      leaf svc-input-bandwidth {
        type uint64;
        units "bps";
        mandatory true;
        description
          "From the customer site's perspective, the
           service input bandwidth of the connection
           or download bandwidth from the SP to
           the site.";
      }
      leaf svc-output-bandwidth {
        type uint64;
        units "bps";
        mandatory true;
        description
          "From the customer site's perspective,
           the service output bandwidth of the
           connection or upload bandwidth from
           the site to the SP.";
      }
      leaf svc-mtu {
        type uint16;
        units "bytes";
        mandatory true;
        description
          "MTU at service level.  If the service is IP,
           it refers to the IP MTU.  If CsC is enabled,
           the requested 'svc-mtu' leaf will refer
            to the MPLS MTU and not to the IP MTU.";
      }
      container qos {
        if-feature "vpn-common:qos";
```

```
container qos-classification-policy {
  list rule {
    key "id";
    ordered-by user;
    leaf id {
      type string;
      description
        "A description identifying the
         qos-classification-policy rule.";
    }
    choice match-type {
      default "match-flow";
      case match-flow {
        choice l3 {
          container ipv4 {
            uses pf:acl-ip-header-fields;
            uses pf:acl-ipv4-header-fields;
            description
              "Rule set that matches IPv4 header.";
          }
          container ipv6 {
            uses pf:acl-ip-header-fields;
            uses pf:acl-ipv6-header-fields;
            description
              "Rule set that matches IPv6 header.";
          }
          description
            "Either IPv4 or IPv6.";
        }
        choice l4 {
          container tcp {
            uses pf:acl-tcp-header-fields;
            uses ports;
            description
              "Rule set that matches TCP header.";
          }
          container udp {
            uses pf:acl-udp-header-fields;
            uses ports;
            description
              "Rule set that matches UDP header.";
          }
          description
            "Can be TCP or UDP";
        }
      }
      case match-application {
        leaf match-application {
```

```
                type identityref {
                  base vpn-common:customer-application;
                }
                description
                  "Defines the application to match.";
              }
            }
            description
              "Choice for classification.";
          }
          leaf target-class-id {
            type string;
            description
              "Identification of the class of service.
               This identifier is internal to the
               administration.";
          }
          description
            "List of marking rules.";
        }
        description
          "Configuration of the traffic classification
           policy.";
      }
      container qos-profile {
        list qos-profile {
          key "profile";
          description
            "QoS profile.
             Can be standard profile or customized
             profile.";
          leaf profile {
            type leafref {
              path "/l3vpn-ntw/vpn-profiles"
                 + "/valid-provider-identifiers"
                 + "/qos-profile-identifier/id";
            }
            description
              "QoS profile to be used.";
          }
          leaf direction {
            type identityref {
              base vpn-common:qos-profile-direction;
            }
            default "vpn-common:both";
            description
              "The direction to which the QoS profile
               is applied.";
```

```
        }
      }
      description
        "QoS profile configuration.";
    }
    description
      "QoS configuration.";
  }
  container carrierscarrier {
    if-feature "vpn-common:carrierscarrier";
    leaf signalling-type {
      type enumeration {
        enum ldp {
          description
            "Use LDP as the signalling protocol
             between the PE and the CE.  In this
             case, an IGP routing protocol must
             also be activated.";
        }
        enum bgp {
          description
            "Use BGP as the signalling protocol
             between the PE and the CE.
             In this case, BGP must also be configured
             as the routing protocol.";
          reference
            "RFC 8277: Using BGP to Bind MPLS Labels
                       to Address Prefixes";
        }
      }
      default "bgp";
      description
        "MPLS signalling type.";
    }
    description
      "This container is used when the customer
       provides MPLS-based services.  This is
       only used in the  case of CsC (i.e., a
       customer builds an MPLSservice using an
       IP VPN to carry its traffic).";
  }
  container multicast {
    if-feature "vpn-common:multicast";
    leaf site-type {
      type enumeration {
        enum receiver-only {
          description
            "The site only has receivers.";
```

```
        }
        enum source-only {
          description
            "The site only has sources.";
        }
        enum source-receiver {
          description
            "The site has both sources and
             receivers.";
        }
      }
      default "source-receiver";
      description
        "Type of multicast site.";
    }
    leaf address-family {
      type vpn-common:address-family;
      description
        "Address family.";
    }
    leaf protocol-type {
      type enumeration {
        enum host {
          description
            "Hosts are directly connected to the
             provider network.

             Host protocols such as IGMP or MLD are
             required.";
        }
        enum router {
          description
            "Hosts are behind a customer router.
             PIM will be implemented.";
        }
        enum both {
          description
            "Some hosts are behind a customer router,
             and some others are directly connected
             to the provider network.  Both host and
             routing protocols must be used.

             Typically,  IGMP and PIM will be
             implemented.";
        }
      }
      default "both";
      description
```

```
                    "Multicast protocol type to be used with
                     the customer site.";
                }
                leaf remote-source {
                  type boolean;
                  default "false";
                  description
                    "When true, there is no PIM adjacency on
                     the interface.";
                }
                description
                  "Multicast parameters for the site.";
              }
              description
                "Service parameters on the attachment.";
            }
            description
              "List of accesses for a site.";
          }
          description
            "List of accesses for a site.";
        }
        container maximum-routes {
          list address-family {
            key "af";
            leaf af {
              type vpn-common:address-family;
              description
                "Indicates the address family
                (IPv4 or IPv6).";
            }
            leaf maximum-routes {
              type uint32;
              description
                "Indicates the maximum prefixes the VRF
                 can accept for this address family.";
            }
            description
              "List of address families.";
          }
          description
            "Defines 'maximum-routes' for the VRF.";
        }
        container multicast {
          if-feature "vpn-common:multicast";
          leaf enabled {
            type boolean;
            default "false";
```

```
        description
          "Enables multicast.";
      }
      leaf-list tree-flavor {
        type identityref {
          base vpn-common:multicast-tree-type;
        }
        description
          "Type of tree to be used.";
      }
      container rp {
        container rp-group-mappings {
          list rp-group-mapping {
            key "id";
            leaf id {
              type uint16;
              description
                "Unique identifier for the mapping.";
            }
            container provider-managed {
              leaf enabled {
                type boolean;
                default "false";
                description
                  "Set to true if the Rendezvous Point (RP)
                   must be a provider-managed node.  Set to
                   false if it is a customer-managed node.";
              }
              leaf rp-redundancy {
                type boolean;
                default "false";
                description
                  "If true, a redundancy mechanism for the
                   RP is required.";
              }
              leaf optimal-traffic-delivery {
                type boolean;
                default "false";
                description
                  "If true, the SP must ensure that
                   traffic uses an optimal path.  An SP may
                   use Anycast RP or RP-tree-to-SPT
                   switchover architectures.";
              }
              container anycast {
                when "../rp-redundancy = 'true' and
                      ../optimal-traffic-delivery = 'true'" {
                  description
```

```
            "Only applicable if
             RP redundancy is
             enabled and delivery through
             optimal path is activated.";
          }
          leaf local-address {
            type inet:ip-address;
            description
              "IP local address for PIM RP.
               Usually, it corresponds to router
               ID or primary address";
          }
          leaf-list rp-set-address {
            type inet:ip-address;
            description
              "Address other RP routers
               that share the same RP IP address.";
          }
          description
            "PIM Anycast-RP parameters.";
        }
        description
          "Parameters for a provider-managed RP.";
      }
      leaf rp-address {
        when "../provider-managed/enabled = 'false'" {
          description
            "Relevant when the RP is not
             provider-managed.";
        }
        type inet:ip-address;
        mandatory true;
        description
          "Defines the address of the RP.
           Used if the RP is customer-managed.";
      }
      container groups {
        list group {
          key "id";
          leaf id {
            type uint16;
            description
              "Identifier for the group.";
          }
          choice group-format {
            mandatory true;
            case group-prefix {
              leaf group-address {
```

```
                    type inet:ip-prefix;
                    description
                      "A single multicast group prefix.";
                  }
                }
                case startend {
                  leaf group-start {
                    type inet:ip-address;
                    description
                      "The first multicast group address in
                       the multicast group address range.";
                  }
                  leaf group-end {
                    type inet:ip-address;
                    description
                      "The last multicast group address in
                       the multicast group address range.";
                  }
                }
                description
                  "Choice for multicast group format.";
              }
              description
                "List of multicast groups.";
            }
            description
              "Multicast groups associated with the RP.";
          }
          description
            "List of RP-to-group mappings.";
        }
        description
          "RP-to-group mappings parameters.";
      }
      container rp-discovery {
        leaf rp-discovery-type {
          type identityref {
            base vpn-common:multicast-rp-discovery-type;
          }
          default "vpn-common:static-rp";
          description
            "Type of RP discovery used.";
        }
        container bsr-candidates {
          when "derived-from-or-self(../rp-discovery-type, "
             + "'vpn-common:bsr-rp')" {
            description
              "Only applicable if discovery type
```

```
                    is BSR-RP.";
                }
                leaf-list bsr-candidate-address {
                  type inet:ip-address;
                  description
                    "Address of candidate Bootstrap Router
                    (BSR).";
                }
                description
                  "Container for List of Customer
                   BSR candidate's addresses.";
              }
              description
                "RP discovery parameters.";
            }
            description
              "RP parameters.";
          }
          container msdp {
            if-feature "msdp";
            leaf enabled {
              type boolean;
              default "false";
              description
                "If true, MSDP is activated.";
            }
            leaf peer {
              type inet:ip-address;
              description
                "IP address of the MSDP peer.";
            }
            leaf local-address {
              type inet:ip-address;
              description
                "IP address of the local end. This local
                 address must be configured on the
                 node.";
            }
            description
              "MSDP parameters.";
          }
          description
            "Multicast global parameters for the VPN
             service.";
        }
        description
          "List for VPN node.";
      }
```

```
        }
        description
          "List of VPN services.";
      }
      description
        "Top-level container for the VPN services.";
    }
    description
      "Main container for L3VPN services management.";
  }
}
<CODE ENDS>
```

Figure 25

9. IANA Considerations

This document requests IANA to register the following URI in the "ns" subregistry within the "IETF XML Registry" [RFC3688]:

```
URI: urn:ietf:params:xml:ns:yang:ietf-l3vpn-ntw
Registrant Contact: The IESG.
XML: N/A; the requested URI is an XML namespace.
```

This document requests IANA to register the following YANG module in the "YANG Module Names" subregistry [RFC6020] within the "YANG Parameters" registry.

```
name: ietf-l3vpn-ntw
namespace: urn:ietf:params:xml:ns:yang:ietf-l3vpn-ntw
maintained by IANA: N
prefix: l3nm
reference: RFC XXXX
```

10. Security Considerations

The YANG module specified in this document defines 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 [RFC8466].

The Network Configuration Access Control Model (NACM) [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.

The "ietf-l3vpn-ntw" module is used to manage Layer 3 VPNs in a service provider backbone network. Hence, the module can be used to request, modify, or retrieve L3VPN services. For example, the creation of a 'vpn-service' leaf instance triggers the creation of an L3VPN Service in a service provider network.

Due to the foreseen use of the "ietf-l3vpn-ntw" module, there are a number of data nodes defined in the 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) and delete operations to these data nodes without proper protection or authentication can have a negative effect on network operations. These are the subtrees and data nodes and their sensitivity/ vulnerability in the "ietf-l3vpn-ntw" module:

- 'vpn-service': An attacker who is able to access network nodes can undertake various attacks, such as deleting a running L3VPN Service, interrupting all the traffic of a client. In addition, an attacker may modify the attributes of a running service (e.g., QoS, bandwidth, routing protocols), leading to malfunctioning of the service and therefore to SLA violations. In addition, an attacker could attempt to create a L3VPN Service or adding a new network access. Such activity can be detected by adequately monitoring and tracking network configuration changes.

Some of the readable data nodes in the "ietf-l3vpn-ntw" 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:

- 'customer-name' and 'ip-connection': An attacker can retrieve privacy-related information which can be used to track a customer. Disclosing such information may be considered as a violation of the customer-provider trust relationship.

The following summarizes the foreseen risks of using the "ietf-l3vpn-ntw" module can be classified into:

- Malicious clients attempting to delete or modify VPN services.

- Unauthorized clients attempting to create/modify/delete a VPN service.

- Unauthorized clients attempting to read VPN service related information.

11. Acknowledgements

Thanks to Adrian Farrel and Miguel Cros for the suggestions on the document. Thanks to Philip Eardlay for the review. Lots of thanks for the discussions on opsawg mailing list and at IETF meeting.

This work was supported in part by the European Commission funded H2020-ICT-2016-2 METRO-HAUL project (G.A. 761727).

12. Contributors

Victor Lopez
Telefonica
Email: victor.lopezalvarez@telefonica.com

Daniel King
Old Dog Consulting
Email: daniel@olddog.co.uk

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

Luay Jalil
Verizon
Email: luay.jalil@verizon.com

Qin Wu
Huawei
Email: bill.wu@huawei.com>

Stephane Litkowski
Cisco
Email: slitkows@cisco.com>

Manuel Julian
Vodafone
Email: manuel-julian.lopez@vodafone.com>

Lucia Oliva Ballega
Telefonica
Email: lucia.olivaballega.ext@telefonica.com>

Erez Segev
ECI Telecom
Email: erez.segev@ecitele.com>

13. References

13.1. Normative References

[I-D.ietf-opsawg-vpn-common]
barguil, s., Dios, O., Boucadair, M., and Q. WU, "A Layer 2/3 VPN Common YANG Model", draft-ietf-opsawg-vpn-common-01 (work in progress), September 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>.

[RFC4110] Callon, R. and M. Suzuki, "A Framework for Layer 3 Provider-Provisioned Virtual Private Networks (PPVPNs)", RFC 4110, DOI 10.17487/RFC4110, July 2005, <https://www.rfc-editor.org/info/rfc4110>.

[RFC4364] Rosen, E. and Y. Rekhter, "BGP/MPLS IP Virtual Private Networks (VPNs)", RFC 4364, DOI 10.17487/RFC4364, February 2006, <https://www.rfc-editor.org/info/rfc4364>.

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

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

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

[RFC6513] Rosen, E., Ed. and R. Aggarwal, Ed., "Multicast in MPLS/BGP IP VPNs", RFC 6513, DOI 10.17487/RFC6513, February 2012, <https://www.rfc-editor.org/info/rfc6513>.

[RFC6514] Aggarwal, R., Rosen, E., Morin, T., and Y. Rekhter, "BGP Encodings and Procedures for Multicast in MPLS/BGP IP VPNs", RFC 6514, DOI 10.17487/RFC6514, February 2012, <https://www.rfc-editor.org/info/rfc6514>.

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

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

[RFC7988] Rosen, E., Ed., Subramanian, K., and Z. Zhang, "Ingress Replication Tunnels in Multicast VPN", RFC 7988, DOI 10.17487/RFC7988, October 2016, <https://www.rfc-editor.org/info/rfc7988>.

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

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

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

[RFC8466] Wen, B., Fioccola, G., Ed., Xie, C., and L. Jalil, "A YANG Data Model for Layer 2 Virtual Private Network (L2VPN) Service Delivery", RFC 8466, DOI 10.17487/RFC8466, October 2018, <https://www.rfc-editor.org/info/rfc8466>.

[RFC8519] Jethanandani, M., Agarwal, S., Huang, L., and D. Blair, "YANG Data Model for Network Access Control Lists (ACLs)", RFC 8519, DOI 10.17487/RFC8519, March 2019, <https://www.rfc-editor.org/info/rfc8519>.

13.2. Informative References

[I-D.evenwu-opsawg-yang-composed-vpn]
Even, R., Bo, W., Wu, Q., and Y. Cheng, "YANG Data Model for Composed VPN Service Delivery", draft-evenwu-opsawg-yang-composed-vpn-03 (work in progress), March 2019.

[I-D.ietf-idr-bgp-model]
Jethanandani, M., Patel, K., Hares, S., and J. Haas, "BGP YANG Model for Service Provider Networks", draft-ietf-idr-bgp-model-09 (work in progress), June 2020.

[I-D.ietf-pim-yang]
Liu, X., McAllister, P., Peter, A., Sivakumar, M., Liu, Y., and f. hu, "A YANG Data Model for Protocol Independent Multicast (PIM)", draft-ietf-pim-yang-17 (work in progress), May 2018.

[I-D.ietf-rtgwg-qos-model]
Choudhary, A., Jethanandani, M., Strahle, N., Aries, E., and I. Chen, "YANG Model for QoS", draft-ietf-rtgwg-qos-model-02 (work in progress), July 2020.

[RFC3618] Fenner, B., Ed. and D. Meyer, Ed., "Multicast Source Discovery Protocol (MSDP)", RFC 3618, DOI 10.17487/RFC3618, October 2003, <https://www.rfc-editor.org/info/rfc3618>.

[RFC3644] Snir, Y., Ramberg, Y., Strassner, J., Cohen, R., and B. Moore, "Policy Quality of Service (QoS) Information Model", RFC 3644, DOI 10.17487/RFC3644, November 2003, <https://www.rfc-editor.org/info/rfc3644>.

[RFC4026] Andersson, L. and T. Madsen, "Provider Provisioned Virtual Private Network (VPN) Terminology", RFC 4026, DOI 10.17487/RFC4026, March 2005, <https://www.rfc-editor.org/info/rfc4026>.

[RFC4176] El Mghazli, Y., Ed., Nadeau, T., Boucadair, M., Chan, K., and A. Gonguet, "Framework for Layer 3 Virtual Private Networks (L3VPN) Operations and Management", RFC 4176, DOI 10.17487/RFC4176, October 2005, <https://www.rfc-editor.org/info/rfc4176>.

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

[RFC5880] Katz, D. and D. Ward, "Bidirectional Forwarding Detection (BFD)", RFC 5880, DOI 10.17487/RFC5880, June 2010, <https://www.rfc-editor.org/info/rfc5880>.

[RFC6037] Rosen, E., Ed., Cai, Y., Ed., and IJ. Wijnands, "Cisco Systems' Solution for Multicast in BGP/MPLS IP VPNs", RFC 6037, DOI 10.17487/RFC6037, October 2010, <https://www.rfc-editor.org/info/rfc6037>.

[RFC6982] Sheffer, Y. and A. Farrel, "Improving Awareness of Running Code: The Implementation Status Section", RFC 6982, DOI 10.17487/RFC6982, July 2013, <https://www.rfc-editor.org/info/rfc6982>.

[RFC7527] Asati, R., Singh, H., Beebee, W., Pignataro, C., Dart, E., and W. George, "Enhanced Duplicate Address Detection", RFC 7527, DOI 10.17487/RFC7527, April 2015, <https://www.rfc-editor.org/info/rfc7527>.

[RFC8277] Rosen, E., "Using BGP to Bind MPLS Labels to Address Prefixes", RFC 8277, DOI 10.17487/RFC8277, October 2017, <https://www.rfc-editor.org/info/rfc8277>.

[RFC8299] Wu, Q., Ed., Litkowski, S., Tomotaki, L., and K. Ogaki, "YANG Data Model for L3VPN Service Delivery", RFC 8299, DOI 10.17487/RFC8299, January 2018, <https://www.rfc-editor.org/info/rfc8299>.

[RFC8309] Wu, Q., Liu, W., and A. Farrel, "Service Models Explained", RFC 8309, DOI 10.17487/RFC8309, January 2018, <https://www.rfc-editor.org/info/rfc8309>.

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

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

[RFC8345] Clemm, A., Medved, J., Varga, R., Bahadur, N., Ananthakrishnan, H., and X. Liu, "A YANG Data Model for Network Topologies", RFC 8345, DOI 10.17487/RFC8345, March 2018, <https://www.rfc-editor.org/info/rfc8345>.

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

[RFC8453] Ceccarelli, D., Ed. and Y. Lee, Ed., "Framework for Abstraction and Control of TE Networks (ACTN)", RFC 8453, DOI 10.17487/RFC8453, August 2018, <https://www.rfc-editor.org/info/rfc8453>.

[RFC8512] Boucadair, M., Ed., Sivakumar, S., Jacquenet, C., Vinapamula, S., and Q. Wu, "A YANG Module for Network Address Translation (NAT) and Network Prefix Translation (NPT)", RFC 8512, DOI 10.17487/RFC8512, January 2019, <https://www.rfc-editor.org/info/rfc8512>.

Appendix A. L3VPN Examples

A.1. 4G VPN Provisioning Example

L3VPNs are widely used to deploy 3G/4G, fixed, and enterprise services mainly because several traffic discrimination policies can be applied within the network to deliver to the mobile customers a service that meets the SLA requirements.

As it is shown in the Figure 26, typically, an eNodeB (CE) is directly connected to the access routers of the mobile backhaul and their logical interfaces (one or many according to the Service type) are configured in a VPN that transports the packets to the mobile core platforms. In this example, a 'vpn-node' is created with two 'vpn-network-accesses'.

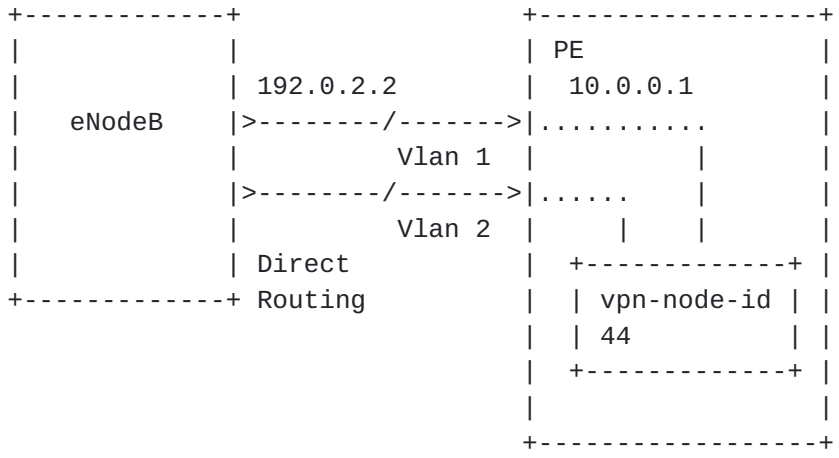


Figure 26: Mobile Backhaul Example

To create a L3VPN service using the L3NM model, the following sample steps can be followed:

First: Create the 4G VPN Service (Figure 27).

```
POST: /restconf/data/ietf-l3vpn-ntw:l3vpn-ntw/vpn-services
Host: example.com
Content-Type: application/yang-data+json

{
  "ietf-l3vpn-ntw:vpn-services": {
    "vpn-service": [
      {
        "vpn-id": "4G",
        "customer-name": "mycustomer",
        "vpn-service-topology": "custom",
        "description": "VPN to deploy 4G services"
      }
    ]
  }
}
```

Figure 27: Create VPN Service

Second: Create a VPN Node as depicted in Figure 28. In this type of service, the VPN Node is equivalent to the VRF configured in the physical device ('ne-id'=10.0.0.1).

```
POST: /restconf/data/ietf-l3vpn-ntw:l3vpn-ntw/\
      vpn-services/vpn-service=4G
Host: example.com
Content-Type: application/yang-data+json

{
  "ietf-l3vpn-ntw:vpn-nodes": {
    "vpn-node": [
      {
        "vpn-node-id": "44",
        "ne-id": "10.0.0.1",
        "local-autonomous-system": "65550",
        "rd": "0:65550:1",
        "vpn-targets": {
          "vpn-target": [
            {
              "id": "1",
              "route-targets": [
                "0:65550:1"
              ],
              "route-target-type": "both"
            }
          ]
        }
      }
    ]
  }
}
```

Figure 28: Create VPN Node

Finally, two VPN Network Accesses are created using the same physical port ('port-id'=1/1/1). Each 'vpn-network-access' has a particular VLAN (1,2) to differentiate the traffic between: Sync and data (Figure 29).

```
POST: /restconf/data/ietf-l3vpn-ntw:l3vpn-ntw/\
      vpn-services/vpn-service=4G/vpn-nodes/vpn-node=44
content-type: application/yang-data+json

{
  "ietf-l3vpn-ntw:vpn-network-accesses": {
    "vpn-network-access": [
      {
        "vpn-network-access-id": "1/1/1.1",
        "port-id": "1/1/1",
        "description": "Interface SYNC to eNODE-B",
        "status": {
```

```
            "admin-enabled": "true"
          },
          "vpn-network-access-type": "vpn-common:point-to-point",
          "ip-connection": {
            "ipv4": {
              "address-allocation-type": "static-address",
              "static-addresses": {
                "primary-address": "1",
                "address": [
                  {
                    "address-id": "1",
                    "s-provider-address": "192.0.2.1",
                    "s-customer-address": "192.0.2.1",
                    "s-prefix-length": 32
                  }
                ]
              }
            }
          },
          "routing-protocols": {
            "routing-protocol": [
              {
                "id": "1",
                "type": "vpn-common:direct"
              }
            ]
          }
        },
        {
          "vpn-network-access-id": "1/1/1.2",
          "port-id": "1/1/1",
          "description": "Interface DATA to eNODE-B",
          "status": {
            "admin-enabled": "true"
          },
          "ip-connection": {
            "ipv4": {
              "address-allocation-type": "static-address",
              "static-addresses": {
                "primary-address": "1",
                "address": [
                  {
                    "address-id": "1",
                    "s-provider-address": "192.0.2.1",
                    "s-customer-address": "192.0.2.2",
                    "s-prefix-length": 32
                  }
                ]
```

```
            }
          }
        },
        "routing-protocols": {
          "routing-protocol": [
            {
              "id": "1",
              "type": "vpn-common:direct"
            }
          ]
        }
      }
    ]
  }
}
```

Figure 29: Create VPN Network Access

A.2. Multicast VPN Provisioning Example

IPTV is mainly distributed through multicast over the LANs. In the following example, PIM-SM is enabled and functional between the PE and the CE. The PE receives multicast traffic from a CE that is directly connected to the multicast source. The signaling between PE and CE is achieved using BGP. Also, RP is statically configured for a multicast group.

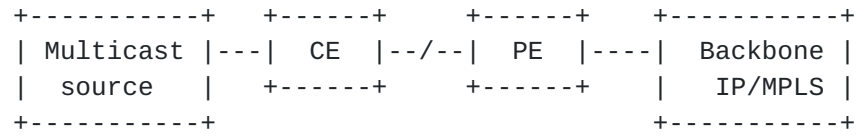


Figure 30: Multicast L3VPN Service Example

To configure a Multicast L3VPN service using the L3NM model the procedure and the JSON with the data structure is the following:

First, the multicast service is created (see the excerpt of the request message body shown in Figure 31)

```
{
  "ietf-l3vpn-ntw:vpn-services": {
    "vpn-service": [
      {
        "vpn-id": "Multicast-IPTV",
        "customer-name": "310",
        "vpn-service-topology": "vpn-common:hub-spoke",
        "description": "Multicast IPTV VPN service"
      }
    ]
  }
}
```

Figure 31: Create Multicast VPN Service (Excerpt of the Message Request Body)

Then, the VPN nodes are created (see the excerpt of the request message body shown in Figure 32). In this example, the VPN Node will represent VRF configured in the physical device.

```
{
  "ietf-l3vpn-ntw:vpn-node": [
    {
      "vpn-node-id": "500003105",
      "ne-id": "10.250.2.202",
      "autonomous-system": "3816",
      "description": "VRF_IPTV_MULTICAST",
      "router-id": "10.250.2.202",
      "address-family": "ipv4",
      "node-role": "vpn-common:hub-role",
      "rd": "3816:31050202",
      "multicast": {
        "enabled": "true",
        "rp": {
          "rp-group-mappings": {
            "rp-group-mapping": [
              {
                "id": "1",
                "rp-address": "172.19.48.17",
                "groups": {
                  "group": [
                    {
                      "id": "1",
                      "group-address": "239.130.0.0/15"
                    }
                  ]
                }
              }
            ]
          },
          "rp-discovery": {
            "rp-discovery-type": "vpn-common:static-rp"
          }
        }
      }
    }
  ]
}
```

Figure 32: Create Multicast VPN Node (Excerpt of the Message Request Body)

Finally, create the VPN Network Access with multicast enabled (see the excerpt of the request message body shown in Figure 33).

```
 {
  "ietf-l3vpn-ntw:vpn-network-access": {
    "vpn-network-access-id": "1/1/1",
```

```
      "description": "Connected_to_source",
      "status": {
        "admin-enabled": "true"
      },
      "vpn-network-access-type": "vpn-common:point-to-point",
      "ip-connection": {
        "ipv4": {
          "address-allocation-type": "static-address",
          "static-addresses": {
            "primary-address": "1",
            "address": [
              {
                "address-id": "1",
                "s-provider-address": "172.19.48.1",
                "s-prefix-length": 30
              }
            ]
          }
        }
      },
      "routing-protocols": {
        "routing-protocol": [
          {
            "id": "1",
            "type": "vpn-common:bgp",
            "bgp": {
              "peer-autonomous-system": "6500",
              "local-autonomous-system": "3816",
              "address-family": "ipv4",
              "neighbor": "172.19.48.2",
              "description": "Connected to CE"
            }
          }
        ]
      },
      "service": {
        "multicast": {
          "multicast-site-type": "source-only",
          "multicast-address-family": {
            "ipv4": "true"
          },
          "protocol-type": "router"
        }
      }
    }
  }
```

Figure 33: Create VPN Network Access (Excerpt of the Message Request Body)

Appendix B. Implementation Status

This section records the status of known implementations of the Yang module defined by this specification at the time of posting of this Internet-Draft, and is based on a proposal described in [RFC6982]. The description of implementations in this section is intended to assist the IETF in its decision processes in progressing drafts to RFCs. Please note that the listing of any individual implementation here does not imply endorsement by the IETF. Furthermore, no effort has been spent to verify the information presented here that was supplied by IETF contributors. This is not intended as, and must not be construed to be, a catalog of available implementations or their features. Readers are advised to note that other implementations may exist.

According to [RFC6982], "this will allow reviewers and working groups to assign due consideration to documents that have the benefit of running code, which may serve as evidence of valuable experimentation and feedback that have made the implemented protocols more mature. It is up to the individual working groups to use this information as they see fit".

Note the RFC Editor: As per [RFC6982] guidelines, please remove this Implementation Status apendix prior publication.

B.1. Nokia Implementation

Details can be found at: https://github.com/IETF-OPSAWG-WG/l3nm/blob/master/Implementattion/Nokia.txt

B.2. Huawei Implementation

Details can be found at: https://github.com/IETF-OPSAWG-WG/l3nm/blob/master/Implementattion/Huawei.txt

B.3. Infinera Implementation

Details can be found at: https://github.com/IETF-OPSAWG-WG/l3nm/blob/master/Implementattion/Infinera.txt

B.4. Ribbon-ECI Implementation

Details can be found at: https://github.com/IETF-OPSAWG-WG/l3nm/blob/master/Implementattion/Ribbon-ECI.txt

Authors' Addresses

Samier Barguil
Telefonica
Madrid
ES

Email: samier.barguilgiraldo.ext@telefonica.com

Oscar Gonzalez de Dios (editor)
Telefonica
Madrid
ES

Email: oscar.gonzalezdedios@telefonica.com

Mohamed Boucadair (editor)
Orange
Rennes 35000
France

Email: mohamed.boucadair@orange.com

Luis Angel Munoz
Vodafone
ES

Email: luis-angel.munoz@vodafone.com

Alejandro Aguado
Nokia
Madrid
ES

Email: alejandro.aguado_martin@nokia.com